Number Theory		
1.	Sieve	2
2.	Number of Divisor	2
3.	All Divisor	2
4.	Big MOD (Iterative)	2
5.	Bitwise Sieve	2
6.	Modular Inverse + Extended Euclid	2
7.	Factorial	2
8.	Leading Digit of Power (n^m)	3
9.	Linear Diphantine Eqn+Extd. Euclid (Iterative)	3
10.	Simple Hyperbolic Diphantine Eqn.	3
11.	Euler Phi / Totient	3
12.	GCD(I, N) <= m	3
13.	Segmented Sieve	4
14.	Sum of Divisor (SOD)	4
15.	Sum of Number of Divisor (SNOD)	4
16.	Sum of Sum of Divisor (SSOD)	4
17.	XOR of 1 to N	4
18.	LCM SUM	4
19.	Matrix Exponential	5
20.	NCR	5
21.	Negative Base	5
22.	Construct N from SOD	5
23.	Derangement	5
24.	String Multiply	5
25.	Wilson Theorem	6
26.	Catalan Number	6
Graph Theory		
27.	Direction Array	6
28.	Articulation Bridge	6
29.	Articulation Point	6
30.	Strongly Connected Component (SCC)	6
31.	Shrink (mamun4122)	7
32.	Lowest Common Ancestor (LCA)	7
33.	LCA + MST	7
34.	Stable Marriage	8
35.	Minimum Vertex Cover	8
36.	Kth Best Shortest Path	9
37.	Djkstra	9
38.	Bellman Ford	9
39.	Floyd Warshall	9
40.	Tree Diameter	9
41.	Farthest node from a Given node	10
42.	Topological Sort	10
43.	Prufer Code to Tree	10
44.	Tree to Prufer Code	10
45.	Euler Circuit	10
46.	Euler Path	10
47.	MST (Kruskal)	10
48.	MST (Prims)	10
49.	Maximum Bipartite Matching	11
50.	2-SAT	11
51.	Erdos & Gallai Theorem	11
52.	Bi-connected Component	12
Dynamic Programming		
53.	LIS (N log K)	12
54.	2d LIS (N log N)	12
55.	LCS (1d)	12
56.	Matrix Chain Multiplication (MCM)	13
57.	Histogram	13
58.	Lonfest Palindrome (Manacher Algo)	13
59.	2d Max Sum (dipta007)	13
60.	2d Max Sum (mamun4122)	14
61.	Coin Change (II)	14
62.	Coin Change (III)	14
63.	Digit DP (dipta007)	14
64.	Digit Dp (mamun4122)	15
65.	Edit Distance	15
Game Theory		
66.	Nim	15
67.	Spurge Grundy Theorem	15
68.	MinMax	16
Data Structure		
69.	Union Find/ Disjoint Set	16
70.	Union Find (mamun4122)	16
71.	Segment Tree	16
72.	Binary Indexed Tree (BIT)	17
73.	Range Minimum Query	17
74.	Prefix Trie	17
String Algorithms		
75.	KMP	18
76.	Z Algorithm	18
77.	Aho Corasick	18
78.	Suffix Array	19
Geometry		
79.	Misc Geometry Formula	21
80.	Misc Trigonometric Func. & Formula	21
81.	Misc Integration Formula	21
82.	Misc Differential Formula	21
83.	Mirror Point of a point W.r.to line	21
84.	Determining if a point lies on the interior of a 3d convex Polygon	21
85.	Misc. Geometry	21
86.	Distance (Point, Point)	22
87.	Distance^2 (Point, Point)	22
88.	Distance (Point, Line)	22
89.	Distance (Point, Segment)	22
90.	IsLeft	22
91.	Intersection (Line, Line)	22
92.	Intersection (Segment, Segment)	22
93.	Perpendicular line of a given line through a Point	22
94.	Area of a 2d-Polygon	22
95.	Point inside Polygon	22
96.	Intersection (Circle, Line)	22
97.	Find Points that are r1 unit away from A, r2 unit away from B	22
98.	Intersection area between 2 circles	22
99.	Circle through the Points	23
100.	Rotating a point anti-clockwise by theta w.r.to origin	23
101.	Convex hull (Graham Scan) O(N log N)	23
102.	Angle between Vectors	23
Misc		
103.	Formula	23
104.	Catalan Number Properties	23
105.	Nth Permutation	23
106.	Backtracking (N queen Problem)	24
107.	Strtok	24
108.	Bitwise Operator	24
109.	Permutation & Combination upto 30	24
110.	Josephus Problem	25
Some Notes		
111.	mamun4122	25
112.	dipta007	25

113.	howcum	25
114.	Palindromic Index	25
115.	Ternary Search	25

NUMBER THEORY:

SIEVE:

```
#define M 1000000
bool marked[M];
vector < int > primes;
void sieve(int n) {
    primes.push_back(2);
    for (int i = 3; i * i <= n; i += 2) {
        if (marked[i] == 0) {
            primes.push_back(i);
            for (int j = i * i; j <= n; j += i + i) {
                marked[j] = 1;
            }
        }
    }
}
```

NUMBER OF DIVISOR:

```
int NOD(int n) {
    int sqrtn = sqrt(n);
    int res = 1;
    for (int i = 0; i < primes.size() && primes[i] <= sqrtn; i++) {
        if (n % primes[i] == 0) {
            int p = 0; /*Counter for power of prime*/
            while (n % primes[i] == 0) {
                n /= primes[i];
                p++;
            }
            sqrtn = sqrt(n);
            p++; /*Increase it by one at end*/
            res *= p; /*Multiply with answer*/
        }
    }
    if (n != 1) {
        res *= 2; /*Remaining prime has power p^1. So multiply with 2*/
    }
    return res;
}
```

ALL DIVISOR:

```
vector < ll > mainfactor;
vector < int > countfactor;
vector < ll > allfactor;
void alldivisor(int idx, ll num) {
    if (idx == mainfactor.size()) {
        allfactor.push_back(num);
        return;
    }
    alldivisor(idx + 1, num);
    //    alldivisor(idx+1,mainfactor[idx]);
    for (int i = countfactor[idx]; i; i--) {
        alldivisor(idx + 1, num * mainfactor[idx]);
        num *= mainfactor[idx];
    }
}
```

BIG MOD: (Iterative)

```
int bigmod(int b, int p, int m) {
    int res = 1 % m, x = b % m;
    while (p) {
        if (p & 1) res = (res * x) % m;
        x = (x * x) % m;
        p >>= 1;
    }
    return res;
}
```

BITWISE SIEVE:

```
#define M 100000000
int marked[M / 64 + 2];
void sieve(int n) {
    for (int i = 3; i * i < n; i += 2) {
        if (!bitCheck(i)) {
            for (int j = i * i; j <= n; j += i + i) {
                bitOn(j);
            }
        }
    }
}
bool isPrime(int num) {
    return num > 1 && (num == 2 || ((num & 1) && !on(num)));
}
```

MODULAR INVERSE + EXTENDED EUCLID:

```
pii extendedEuclid(int a, int b) {
    if (b == 0)
        return pii(1, 0);
    else {
        pii d = extendedEuclid(b, a % b);
        return pii(d.ss, d.ff - d.ss * (a / b));
    }
}
int modularInverse(int a, int n) { // returns a er modular Inverse ; n dara mod kore
    pii ret = extendedEuclid(a, n);
    return ((ret.ff % n) + n) % n;
}
```

FACTORIAL:

```
///Digits of N! in Different Base
int factorialDigitExtended(int n, int base) {
        double x = 0;
        for (int i = 1; i <= n; i++) {
            x += log10(i) / log10(base); ///Base Conversion
        }
        int res = ((int) x) + 1;
        return res;
    }
///Prime Factorization of Factorial
void factFactorize(int n) {
        for (int i = 0; i < primes.size() && primes[i] <= n; i++) {
            int x = n;
            int freq = 0;
            while (x / primes[i]) {
                freq += x / primes[i];
                x = x / primes[i];
            }
            printf("%d^%d¥n", primes[i], freq);
        }
    }
///leading digits in factorial
///Find the first K digits of N!
///k=first k digits
int leadingDigitFact(int n, int k) {
        double fact = 0;
        ///Find log(N!)
        for (int i = 1; i <= n; i++) {
            fact += log10(i);
        }
        ///Find the value of q
        double q = fact - floor(fact + EPS);
        double B = pow(10, q);
        ///Shift decimal point k-1 times
        for (int i = 0; i < k - 1; i++) {
            B *= 10;
        }
        ///Don't forget to floor it
        return floor(B + eps);
    }
///last digit of factorial
int last_digit_factorial(int N) {
    int i, j, ans = 1, a2 = 0, a5 = 0, a;

    for (i = 1; i <= N; i++) {
        j = i;
        //divide i by 2 and 5
        while (j % 2 == 0) {
            j /= 2;
            a2++;
```

```
        }
        while (j % 5 == 0) {
            j /= 5;
            a5++;
        }
        ans = (ans * (j % 10)) % 10;
    }
    a = a2 - a5;
    for (i = 1; i <= a; i++)
        ans = (ans * 2) % 10;

    return ans;
}
```

LEADING DIGIT OF POWER: (N^M)

```
///leading digits of n^k
ll leadingdigit(ll n, ll k, ll dig) {
    ///log10(x)=y
    double y = (double) k * log10(n);
    ///if y=123.456 we are setting y=0.456
    y = y - floor(y);
    ///we are getting 10^y for reverse processing
    y = pow(10, y);
    ///now we are getting the digits by shifting the decimal to right
    rep(i, dig - 1) y *= 10;
    return (ll) floor(y);
}
```

LINEAR DIOPHANTINE EQN + EXTENDED EUCLID (ITERATIVE):

```
int ext_gcd(int A, int B, int * X, int * Y) {
    int x2, y2, x1, y1, x, y, r2, r1, q, r;
    x2 = 1;  y2 = 0;
    x1 = 0;  y1 = 1;
    for (r2 = A, r1 = B; r1 != 0; r2 = r1, r1 = r, x2 = x1, y2 = y1, x1 = x, y1 = y) {
        q = r2 / r1;
        r = r2 % r1;
        x = x2 - (q * x1);
        y = y2 - (q * y1);
    } * X = x2; * Y = y2;
    return r2;
}
bool linearDiophantine(int A, int B, int C, int * x, int * y) {
    int g = gcd(A, B);
    if (C % g != 0) return false; //No Solution
    int a = A / g, b = B / g, c = C / g;
    ext_gcd(a, b, x, y); //Solve ax + by = 1
    if (g < 0) { //Make Sure gcd(a,b) = 1
        a *= -1;
        b *= -1;
        c *= -1;
    } * x *= c; * y *= c; //ax + by = c
    return true; //Solution Exists
}
int main() {
    int x, y, A = 2, B = 3, C = 5;
    bool res = linearDiophantine(A, B, C, & x, & y);
    if (res == false) printf("No Solution¥n");
    else {
        printf("One Possible Solution (%d %d) ¥n", x, y);
        int g = gcd(A, B);
        int k = 1; //Use different value of k to get different solutions
        printf("Another Possible Solution (%d %d)¥n", x + k * (B / g), y - k * (A / g));
    }
}
```

Simple Hyperbolic Diophantine Equation:

```
bool isValidSolution(int a, int b, int c, int p, int div) {
    if (((div - c) % a) != 0) return false; //x = (div - c) / a
    if (((p - b * div) % (a * div)) != 0) return false; // y = (p-b*div) /(a*div)
    return true;
}
int hyperbolicDiophantine(int a, int b, int c, int d) {
    int p = a * d + b * c;
    if (p == 0) { //ad + bc = 0
        if (-c % a == 0) return -1; //Infinite solutions (-c/a, k)
        else if (-b % a == 0) return -1; //Infinite solutions (k, -b/a)
        else return 0; //No solution
    } else {
        int res = 0;
        //For each divisor of p
        int sqrtn = sqrt(p), div;
        for (int i = 1; i <= sqrtn; i++) {
            if (p % i == 0) { //i is a divisor

                //Check if divisors i,-i,p/i,-p/i produces valid solutions
                if (isValidSolution(a, b, c, p, i)) res++;
                if (isValidSolution(a, b, c, p, -i)) res++;
                if (p / i != i) { //Check whether p/i is different divisor than i
                    if (isValidSolution(a, b, c, p, p / i)) res++;
                    if (isValidSolution(a, b, c, p, -p / i)) res++;
                }
            }
        }
        return res;
    }
}
```

EULER PHI/TOTIENT:

```
int phi(int n) // Oyler er Tochient Function
    {
        int ret = n;
        for (int i = 2; i * i <= n; i++) {
            if (n % i == 0) {
                while (n % i == 0) {
                    n /= i;
                }
                ret -= ret / i;
            }
        }
        if (n > 1) ret = ret - (ret / n);
        return ret;
    }
    ///another if this method needs to be called several times
#define M 1000005
int phi[M];
void calculatePhi() {
    for (int i = 1; i < M; i++) {
        phi[i] = i;
    }
    for (int p = 2; p < M; p++) {
        if (phi[p] == p) { // p is a prime
            for (int k = p; k < M; k += p) {
                phi[k] -= phi[k] / p;
            }
        }
    }
}
```

GCD(I,N) <= M :

```
int main() {
    calculatePhi();
    sieve(M - 1);
    int t;
    getI(t);
    rep(cs, t) {
        ll n, m;
```

```
        int q;
        CLR(cum);
        getL(n);
        getI(q);
        ///getting the prime factor of n
        divisor(n);
        ///getting all the factor of n
        alldivisor(0, 1);
        sort(ALL(allfactor));
        ///generating phi value for all the factor
        repI(i, allfactor.size()) {
            ans.push_back(make_pair(allfactor[i], eulerPhi(n / allfactor[i])));
        }
        printf("Case %d¥n", cs);
        int sz = ans.size();
        ///generating cumalitive sum of phi value
        repI(i, sz) {
            if (i) cum[i] = cum[i - 1] + ans[i].ss;
            else cum[i] = ans[i].ss;
        }
        rep(i, q) {
            getL(m);
            ///binary searching the answer based on the gcd see khata for explaination
            int low = 0, high = sz - 1, flag = -1;
            while (low <= high) {
                int mid = (low + high) / 2;
                if (ans[mid].ff <= m) {
                    flag = mid;
                    low = mid + 1;
                } else high = mid - 1;
            }
            ///flag==-1 means m is negative or zero
            printf("%lld¥n", flag == -1 ? 0 : cum[flag]);
        }
    }
    ///GCD(i,n)<=m
    ///koita i ase jader n er sathe gcd m er cheya choto or equal
    ///q = query
}
```

SEGMENTED SIEVE:

```
#define SIZE 1000005
int arr[SIZE];
int segmentedSieve(int a, int b) {
    if (a == 1) a++;
    int sqrtn = sqrt(b);
    CLR(arr);
    for (int i = 0; i < primes.size() && primes[i] <= sqrtn; i++) {
        int p = primes[i];
        int j = p * p;
        ///If j is smaller than a, then shift it inside of segment [a,b]
        if (j < a) j = ((a + p - 1) / p) * p;
        for (; j <= b; j += p) {
            arr[j - a] = 1; ///mark them as not prime
        }
    }
    int res = 0;
    for (int i = a; i <= b; i++) {
        ///If it is not marked, then it is a prime
        if (arr[i - a] == 0) res++;
    }
    return res;
}
```

Sum of Divisor (SOD):

```
int SOD(int n) {
        int res = 1;
        int sqrtn = sqrt(n);
        for (int i = 0; i < primes.size() && primes[i] <= sqrtn; i++) {
            if (n % primes[i] == 0) {
                int tempSum = 1; //Contains value of (p^0+p^1+...p^a)
                int p = 1;
                while (n % primes[i] == 0) {
                    n /= primes[i];
                    p *= primes[i];
                    tempSum += p;
                }
                sqrtn = sqrt(n);
                res *= tempSum;
            }
        }
        if (n != 1) {
            res *= (n + 1); //Need to multiply (p^0+p^1)
        }
        return res;
    }
    ///SOD(N)=(p01+p11+p21...pa11)×(p02+p12+p22...pa22)×...(p0k+p1k+p2k...pakk)
```

SUM OF NUMBER OF DIVISOR (SNOD):

```
int SNOD(int n) {
    int res = 0;
    int u = sqrt(n);
    for (int i = 1; i <= u; i++) {
        res += (n / i) - i; //Step 1
    }
    res *= 2; //Step 2
    res += u; //Step 3
    return res;
}
```

SUM OF SUM OF DIIVISOR (SSOD):

```
ll ssod(ll n) {
    ll ans = 0;
    for (ll i = 2; i * i <= n; i++) {
        ll j = n / i;
        ans += (i + j) * (j - i + 1) / 2;
        ans += i * (j - i);
    }
    return ans;
}
```

XOR 1 TO N:

```
ll f(long long a) {
    long long res[] = {a,1,a + 1,0};
    return res[a % 4];
}
ll getXor(long long a, long long b) {
    return f(b) ^ f(a - 1);
}
```

LCM SUM:

```
/*Given n, calculate the sum LCM(1,n) + LCM(2,n) + .. + LCM(n,n)*/
ll res[1000010];
ll phi[1000010];
void precal(int n) {
    ///Calculate phi from 1 to n using sieve
    FOR(i, 1, n) phi[i] = i;
    FOR(i, 2, n) {
            if (phi[i] == i) {
                for (int j = i; j <= n; j += i) {
                    phi[j] /= i;
                    phi[j] *= i - 1;
                }
            }
        }
        ///Calculate partial result using sieve
        ///For each divisor d of n, add phi(d)*d to result array
    FOR(i, 1, n) {
        for (int j = i; j <= n; j += i) {
            res[j] += (i * phi[i]);
        }
```

```
    }
}
int main() {
    precal(1000000);
    int kase;
    scanf("%d", &kase);
    while (kase--) {
        ll n;
        getL(n);
        ///We already have partial result in res[n]
        ll ans = res[n] + 1;
        ans *= n;
        ans /= 2;
        printf("%lld¥n", ans);
    }
}
```

MATRIX EXPONENTIAL:

```
struct matrix {
    int v[5][5];
    int row, col; // number of row and column
};
int mod = 10000;
// multiplies two matrices and returns the result
matrix multiply(matrix a, matrix b) {
        assert(a.col == b.row);
        matrix r;
        r.row = a.row;
        r.col = b.col;
        for (int i = 0; i < r.row; i++) {
            for (int j = 0; j < r.col; j++) {
                int sum = 0;
                for (int k = 0; k < a.col; k++) {
                    sum += a.v[i][k] * b.v[k][j];
                    sum %= mod;
                }
                r.v[i][j] = sum;
            }
        }
        return r;
    }
    // returns mat^p
matrix power(matrix mat, int p) {
    assert(p >= 1);
    if (p == 1) return mat;
    if (p % 2 == 1)
        return multiply(mat, power(mat, p - 1));
    matrix ret = power(mat, p / 2);
    ret = multiply(ret, ret);
    return ret;
}
int main() {
    int tcase;
    int a, b, n, m;
    cin >> tcase;
    while (tcase--) {
        // input routine
        cin >> a >> b >> n >> m;
        // preparing the matrix
        matrix mat;
        mat.row = mat.col = 2;
        mat.v[0][0] = mat.v[0][1] = mat.v[1][0] = 1;
        mat.v[1][1] = 0;
        // preparing mod value
        mod = 1;
        for (int i = 0; i < m; i++) mod *= 10;
        a %= mod, b %= mod;
        if (n < 3) {
            if (n == 0) cout << a << endl;
            if (n == 1) cout << b << endl;
            if (n == 2) cout << (a + b) % mod << endl;
        } else {
            mat = power(mat, n - 1);
            int ans = b * mat.v[0][0] + a * mat.v[0][1];
            ans %= mod;
            cout << ans << endl;
        }
    }
}
```

NCR:

```
ncr[0][0] = 1;
int limncr = 10;
FOR(i, 1, limncr)
    FOR(j, 0, limncr) {
        if (j > i) ncr[i][j] = 0;
        else if (j == i || j == 0) ncr[i][j] = 1;
        else ncr[i][j] = ncr[i - 1][j - 1] + ncr[i - 1][j];
    }
```

NEGATIVE BASE:

```
string negaBase(int n, int b) {
    int i, tmp;
    string a;
    for (i = 0; n; i++) {
        tmp = n % b;
        n = n / b;
        if (tmp < 0) {
            tmp += (-b), n++;
        }
        a += '0' + tmp;
    }
    for (n = 0; n < (i / 2); n++) swap(a[n], a[i - n - 1]);
    if (i) return a;
    return "0";
}
```

CONSTRUCT N FROM SOD:

```
// powi64(a, b) computes a^b, rememver that prime upto i-1 are used
i64 table[NN + 1][NN + 1]; // if there is an overflow, table[i][j] = inf;
void preprocessTable() {
    for (int i = 0; i <= NN; i++) table[0][i] = 1;
    for (int i = 1; i <= NN; i++) {
        table[i][0] = 1;
        for (int j = 1; j < NN; j++) table[i][j] = table[i][j - 1] + powi64(pr[i - 1], j);
    }
}
vector < i64 > calculateXFromSumOfDivisors(int sum) {
    vector < i64 > res;
    i64 val = 1, prevD = 1;
    for (int i = NN;; i--) {
        if (sum == 1) {
            res.push_back(val); // Here the value is saved
            sum *= prevD, val = 1;
        }
        if (i <= 0 || sum == 1) break;
        for (int j = NN - 1; j >= 0; j--) {
            if (table[i][j] > 1 && (sum % table[i][j] == 0)) {
                val *= powi64(pr[i - 1], j);
                sum /= table[i][j], prevD = table[i][j];
                break;
            }
        }
    }
    return res;
}
```

DERANGEMENT:

```
/*d(n)=(n-1)*(d(n-1)+d(n-2))d(n)=(n-1)*(d(n-1)+d(n-2))
বেস কেস: d(1)=0,d(2)=1*/
```

STRING MULTIPLY:

```
string multiply(string a, int b) {
    // a contains the biginteger in reversed form
    int carry = 0;
    for (int i = 0; i < a.size(); i++) {
        carry += (a[i] - 48) * b;
        a[i] = (carry % 10 + 48);
        carry /= 10;
```

```
    }
    while (carry) {
        a += (carry % 10 + 48);
        carry /= 10;
    }
    return a;
}
```

WILSON THEOREM:

Wilson's theorem states that a natural number n>1 is a prime number if and only if (n−1)!≡−1 (mod n).
This asserts that (n−1)! is exactly 1 less than a multiple of n when n is prime.

If N is a composite number (except for 1 and 4), then (N−1)! ≡0 (mod N)

CATALAN NUMBER:

$$C_n = \frac{2n!}{n!(n+1)!} \qquad C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

GRAPH THEORY:

DIRECTION ARRAY:

```
// 4 direction
int dx[]={-1,1,0,0};
int dy[]={0,0,-1,1};
// 8 direction
int dx[]={-1,1,0,0,-1,-1,1,1};
int dy[]={0,0,-1,1,-1,1,1,-1};
// horse
int dx[] = {-2,-2,2,2,-1,-1,1,1};
int dy[] = {1,-1,-1,1,2,-2,-2,2};
```

ARTICULATION BRIDGE:

```
#define BRIDGENODE 10010
class BridgeFinding {
    int disc[BRIDGENODE];
    int low[BRIDGENODE];
    int col[BRIDGENODE];
    int cnt; ///Timer
    int cc; ///Color
    void tarjan(int s, int parentEdge) {
        disc[s] = low[s] = cnt++;
        col[s] = cc + 1;
        for (int i = 0; i < adj[s].size(); ++i) {
            int t = adj[s][i].ff;
            int edgeNumber = adj[s][i].ss;
            if (edgeNumber == parentEdge) continue;
            if (col[t] <= cc) { ///New node. Discovery.
                tarjan(t, edgeNumber);
                low[s] = min(low[s], low[t]); ///Update back edge extension for S
                if (low[t] > disc[s]) { ///Back edge of T did not go above S
                    ///This edge is Bridge
                }
            } else if (col[t] == cc + 1) { ///Back Edge
                low[s] = min(low[s], disc[t]);
            }
        }
    }
    public:
        vector < pair < int, int > > adj[BRIDGENODE]; ///Enter target and edge number as pair
    void clear(int n) {
        cc += 3; ///cc is now 0. cc+1 is 1
        for (int i = 0; i <= n; i++) {
            adj[i].clear();
        }
    }
    void findBridge(int n, int start = 0) {
        for (int i = start; i <= n; i++) {
            if (col[i] <= cc) {
                tarjan(i, -1);
            }
        }
    }
}bridge;
```

ARTICULATION POINT:

```
#define ARTNODE 10010
class ArticulationPoint {
    int disc[ARTNODE], low[ARTNODE], col[ARTNODE];
    int cnt; ///Timer
    int cc; ///Color
    int root; ///Root of tree
    void tarjan(int s, int p) {
        disc[s] = low[s] = cnt++;
        col[s] = cc + 1;
        int child = 0; ///Needed for root only
        int art = 0;
        for (int i = 0; i < adj[s].size(); ++i) {
            int t = adj[s][i];
            if (t == p) continue; ///Don't go to parent
            if (col[t] <= cc) { ///New node. Discovery.
                child++;
                tarjan(t, s);
                low[s] = min(low[s], low[t]); ///Update back edge extension for S
                if (low[t] >= disc[s]) { ///Back edge of T did not go above S
                    art++; ///S is articulation point for T
                }
            } else if (col[t] == cc + 1) { ///Back Edge
                low[s] = min(low[s], disc[t]);
            }
        }
        if ((s == root && child > 1) || (s != root && art)) {

            ///Edit in this block
            printf("This is a articulation point: %d¥n", s);

        }
    }
    public:
        vector < int > adj[ARTNODE];
    void clear(int n) {
        cc += 3; ///cc is now 0. cc+1 is 1
        for (int i = 0; i <= n; i++) {
            adj[i].clear();
        }
    }
    void findArt(int n, int start = 0) {
        for (int i = start; i <= n; i++) {
            if (col[i] <= cc) {
                root = i;
                tarjan(i, -1);
            }
        }
    }
} art;
///remaining component after removing x
int compo = 0;
if ((s == root && child > 1) || (s != root && art)) {
    if (s == root) compo = child;
    else compo = art + 1;
} else { ///s is not articulation point
    if (p != -1 || adj[s].size()) compo = 1; ///It is not singleton
}
```

Strongly Connected Component (SCC) :

```
//Cycle contains which scc node belongs too.
struct SCC {
    int num[NODE], low[NODE], col[NODE], cycle[NODE], st[NODE];
    int tail, cnt, cc;
    vi adj[NODE];
    SCC(): tail(0), cnt(0), cc(0) {}
    void clear() {
        cc += 3;
        FOR(i, 0, NODE - 1) adj[i].clear();
        tail = 0;
```

```
    }
    void tarjan(int s) {
        num[s] = low[s] = cnt++;
        col[s] = cc + 1;
        st[tail++] = s;
        FOR(i, 0, SZ(adj[s]) - 1) {
            int t = adj[s][i];
            if (col[t] <= cc) {
                tarjan(t);
                low[s] = MIN(low[s], low[t]);
            }
            /*Back edge*/
            else if (col[t] == cc + 1)
                low[s] = MIN(low[s], low[t]);
        }
        if (low[s] == num[s]) {
            while (1) {
                int temp = st[tail - 1];
                tail--;
                col[temp] = cc + 2;
                cycle[temp] = s;
                if (s == temp) break;
            }
        }
    }
    void shrink(int n) {
        FOR(i, 0, n) {
            FOR(j, 0, SZ(adj[i]) - 1) {
                adj[i][j] = cycle[adj[i][j]]; ///Careful. Thi
s will create self-loop. Just ignore i-
>i edges when processing.
            }
        }
        FOR(i, 0, n) {
            if (cycle[i] == i) continue;
            int u = cycle[i];
            FOR(j, 0, SZ(adj[i]) - 1) {
                int v = adj[i][j];
                adj[u].pb(v);
            }
            adj[i].clear();
        }
        FOR(i, 0, n) { ///Not always necessary
            sort(ALL(adj[i]));
            UNIQUE(adj[i]);
        }
    }
    void findSCC(int n) {
        FOR(i, 0, n) {
            if (col[i] <= cc) {
                tarjan(i);
            }
        }
    }
};
```

SHRINK : (mamun4122)

```
bool shrink(int node) {
    repI(i, node) {
        repI(j, adj[i].size()) {
            if (cycle[i] != cycle[adj[i][j]]) {
                scc[cycle[i]].push_back(cycle[adj[i][j]]);
            }
        }
    }
    ///this inserts same edge multiple times
}
```

Lowest Common Ancesstor (LCA):

```
#define mx 100002
int depth[mx]; //লেভেল
int parent[mx][22]; //স্পার্স টেবিল
int T[mx]; //প্যারেন্ট
vector < int > g[mx];
void dfs(int from, int u, int dep) {
    T[u] = from;
    depth[u] = dep;
    for (int i = 0; i < (int) g[u].size(); i++) {
        int v = g[u][i];
        if (v == from) continue;
        dfs(u, v, dep + 1);
    }
}
int lca_query(int N,int p,int q)//N=নোড সংখ্যা
    {
        int tmp, log, i;
        if (depth[p] < depth[q])
            tmp = p, p = q, q = tmp;
        log = 1;
        while (1) {
            int next = log + 1;
        if ((1 << next) > depth[p]) break;
            log++;
        }
        for (i = log; i >= 0; i--)
      if (depth[p] - (1 << i) >= depth[q])
                p = parent[p][i];
        if (p == q)
            return p;
        for (i = log; i >= 0; i--)
         if (parent[p][i] != -
1 && parent[p][i] != parent[q][i])
                p = parent[p][i], q = parent[q][i];
        return T[p];
    }
void lca_init(int N) {
    memset(parent, 1, sizeof(parent)); //শুরুতে সবগুলো ঘরে
-১ থাকবে
    int i, j;
    for (i = 0; i < N; i++)
        parent[i][0] = T[i];
    //can be modified here by looping only to depth from dfs

    for (j = 1; 1 << j < N; j++)
        for (i = 0; i < N; i++)
            if (parent[i][j - 1] != -1)
                parent[i][j] = parent[parent[i][j - 1]][j - 1
];
}
int main(void) {
    g[0].pb(1);g[0].pb(2);g[2].pb(3);g[2].pb(4);
    dfs(0, 0, 0);
    lca_init(5);
    printf("%d¥n", lca_query(5, 3, 4));
    return 0;
}
```

LCA + MST :

```
#define MAXN 200005
struct edge {
    int u, v, pos;
    ll w;
    bool operator < (const edge & p) const {
        return w < p.w;
    }
};
int pr[MAXN];
vector < edge > e;
int find(int r) {
    if (pr[r] == r) return r;
    return pr[r] = find(pr[r]);
}
vector < pair < int, ll > > g[MAXN];
ll ans[MAXN];
///LCA HERE
int depth[MAXN];
int parent[MAXN][30];
int T[MAXN];
ll dist[MAXN][30];
```

```
void dfs(int from, int u, int dep) {
    T[u] = from;
    depth[u] = dep;
    for (int i = 0; i < (int) g[u].size(); i++) {
        int v = g[u][i].ff;
        if (v == from) continue;
        dist[v][0] = g[u][i].ss;
        dfs(u, v, dep + 1);
    }
}
void lca_init(int N) {
    memset(parent, -1, sizeof(parent));
    int i, j;
    for (i = 1; i <= N; i++)
        parent[i][0] = T[i];
    //can be modified here by looping only to depth from dfs

    for (j = 1; 1 << j < N; j++)
        for (i = 1; i <= N; i++)
            if (parent[i][j - 1] != -1) {
                dist[i][j] = max(dist[i][j - 1], dist[parent[
i][j - 1]][j - 1]);
                parent[i][j] = parent[parent[i][j - 1]][j - 1
];
            }
}
ll lca_query(int N, int p, int q) //N=???? ??????
    {
        int tmp, log, i;
        if (depth[p] < depth[q]) swap(p, q);
        ll tmpans = 0;
        log = 1;
        while (1) {
            int next = log + 1;
            if ((1 << next) > depth[p]) break;
            log++;
        }
        for (i = log; i >= 0; i--)
            if (depth[p] - (1 << i) >= depth[q]) {
                tmpans = max(tmpans, dist[p][i]);
                p = parent[p][i];
            }
        if (p == q)
            return tmpans;
        for (i = log; i >= 0; i--)
            if (parent[p][i] != -
1 && parent[p][i] != parent[q][i]) {
                tmpans = max(tmpans, dist[p][i]);
                tmpans = max(tmpans, dist[q][i]);
                p = parent[p][i];
                q = parent[q][i];
            }
        tmpans = max(tmpans, dist[p][0]);
        tmpans = max(tmpans, dist[q][0]);
        return tmpans;
    }
    ///LCA END
int main() {
    int n, m;
    getII(n, m);
    edge get;
    rep(i, m) {
        getII(get.u, get.v);
        getL(get.w);
        get.pos = i;
        e.push_back(get);
    }
    CLR(ans);
    ///mst here
    ll mst = 0;
    sort(e.begin(), e.end());
    for (int i = 1; i <= n; i++) pr[i] = i;
    int count = 0;
    for (int i = 0; i < (int) e.size(); i++) {
        int u = find(e[i].u);
        int v = find(e[i].v);
        if (u != v) {
            pr[u] = v;
            count++;
            mst += e[i].w;
            ans[e[i].pos] = 1;
            g[e[i].u].push_back(make_pair(e[i].v, e[i].w));
            g[e[i].v].push_back(make_pair(e[i].u, e[i].w));
            if (count == n - 1) break;
        }
    }
    dfs(-1, 1, 1);
    lca_init(n);
    repI(i, m) {
        if (ans[e[i].pos]) ans[e[i].pos] = mst;
        else {
            ans[e[i].pos] = mst - lca_query(n, e[i].u, e[i].v
);
            ans[e[i].pos] += e[i].w;
        }
    }
    rep(i, m) printf("%lld¥n", ans[i]);
}
```

STABLE MARRIAGE:

```
/* A person has an integer preference for each of the persons
of the opposite
 * sex, produces a matching of each man to some woman. The ma
tching will follow:
 *          - Each man is assigned to a different woman (n mu
st be at least m)
 *          - No two couples M1W1 and M2W2 will be unstable.
 * Two couples are unstable if (M1 prefers W2 over W1 and W1
prefers M2 over M1)
 * INPUT: m - number of man, n - number of woman (must be at
least as large as m)
 *          - L[i][]: the list of women in order of decreasin
g preference of man i
 *          - R[j][i]: the attractiveness of i to j.
 * OUTPUTS: - L2R[]: the mate of man i (always between 0 and
n-1)
 *          - R2L[]: the mate of woman j (or -
1 if single)      */
/*While there is a free man m: let w be the most-
preferred woman to whom he has not yet proposed, and propose
m to w. If w is free, or is engaged to someone whom she prefe
rs less than m, match m with w, else deny proposal.*/
int m, n, L[MAXM][MAXW], R[MAXW][MAXM], L2R[MAXM], R2L[MAXW],
p[MAXM];
void stableMarriage() {
    memset(R2L, -1, sizeof(R2L));
    memset(p, 0, sizeof(p));
    for (int i = 0; i < m; i++) { // Each man proposes...
        int man = i;
        while (man >= 0) {
            int wom;
            while (1) {
                wom = L[man][p[man]++];
                if (R2L[wom] < 0 || R[wom][man] > R[wom][R2L[
wom]]) break;
            }
            int hubby = R2L[wom];
            R2L[L2R[man] = wom] = man;
            man = hubby;
        }
    }
}
```

MINIMUM VERTEX COVER:

```
#define MAXN 100002
int dp[MAXN][5];
int par[MAXN];
vectoredges[MAXN];
```

```
int f(int u, int isGuard) {
    if (edges[u].size() == 0)
        return 0;
    if (dp[u][isGuard] != -1)
        return dp[u][isGuard];
    int sum = 0;
    for (int i = 0; i < (int) edges[u].size(); i++) {
        int v = edges[u][i];
        if (v != par[u]) {
            par[v] = u;
            if (isGuard == 0)
                sum += f(v, 1);
            else
                sum += min(f(v, 1), f(v, 0));
        }
    }
    return dp[u][isGuard] = sum + isGuard;
}
int main() {
    memset(dp, -1, sizeof(dp));
    int n;
    scanf("%d", & n);
    for (int i = 1; i < n; i++) {
        int u, v;
        scanf("%d%d", & u, & v);
        edges[u].push_back(v);
        edges[v].push_back(u);
    }
    int ans = 0;
    ans = min(f(1, 1), f(1, 0));
    printf("%d¥n";, ans);
    return 0;
}
```

KTH BEST SHORTEST PATH:

```
int m, n, deg[MM], source, sink, K, val[MM][12];
struct edge {
    int v, w;
}
adj[MM][500];
struct info {
    int v, w, k;
    bool operator < (const info & b) const {
        return w > b.w;
    }
};
priority_queue < info, vector < info > > Q;
void kthBestShortestPath() {
    int i, j;
    info u, v;
    for (i = 0; i < n; i++)
        for (j = 0; j < K; j++) val[i][j] = inf;
    u.v = source;
    u.k = 0;
    u.w = 0;
    Q.push(u);
    while (!Q.empty()) {
        u = Q.top();
        Q.pop();
        for (i = 0; i < deg[u.v]; i++) {
            v.v = adj[u.v][i].v;
            int cost = adj[u.v][i].w + u.w;
            for (v.k = u.k; v.k < K; v.k++) {
                if (cost == inf) break;
                if (val[v.v][v.k] > cost) {
                    swap(cost, val[v.v][v.k]);
                    v.w = val[v.v][v.k];
                    Q.push(v);
                    break;
                }
            }
            for (v.k++; v.k < K; v.k++) {
                if (cost == inf) break;
                if (val[v.v][v.k] > cost) swap(cost, val[v.v][v.k]);
                }
            }
        }
    }
```

DIJKSTRA:

```
#define mx 100005
vector < int > adj[mx], cost[mx];
struct node {
    int u, w;
    node(int a, int b) { u = a;w = b;}
    bool operator < (const node & p) const {
        return w > p.w;
    }
};
int dist[mx], par[mx];
int dijkstra(int dest) {
    MEM(dist, 63);
    SET(par);
    priority_queue < node > q;
    q.push(node(1, 0));
    dist[1] = 0;
    while (!q.empty()) {
        node top = q.top();
        q.pop();
        int u = top.u;
        if (u == dest) return dist[dest];
        repI(i, adj[u].size()) {
            int v = adj[u][i];
            if (dist[u] + cost[u][i] < dist[v]) {
                dist[v] = dist[u] + cost[u][i];
                par[v] = u;
                q.push(node(v, dist[v]));
            }
        }
    }
    return -1;
}
```

BELLMAN FORD:

```
#define MAXE 10005
#define MAXN 105
int dist[MAXN], edge_u[MAXE], edge_v[MAXE], edge_cost[MAXE];
int main() {
    int n, m;
    getII(n, m);
    MEM(dist, 63);
    dist[1] = 0;
    rep(i, m) getIII(edge_u[i], edge_v[i], edge_cost[i]);
    int neg_cycle = false;
    rep(step, n) {
        int updated = false;
        rep(i, m) {
            int u = edge_u[i], v = edge_v[i];
            if (dist[u] + edge_cost[i] < dist[v]) {
                updated = true;
                if (step == n) neg_cycle = true;
                dist[v] = dist[u] + edge_cost[i];
            }
        }
        if (updated == false) break;
    }
    if (neg_cycle == false) {
        rep(i, n) cout << dist[i] << endl;
    } else puts("Negative Cycle");
}
```

FLOYD WARSHALL:

```
int d[100][100]; // d[i][j] = distance from i to j
int midMan[100][100] //first set -1
rep(k, n) rep(i, n) rep(j, n)
if (d[i][j] > d[i][k] + d[k][j]) {
    d[i][j] = d[i][k] + d[k][j];
    next[i][j] = next[i][k];
```

```
}
// at 1st in all position = INF
// After floyd if diagonal has <INF -> cycle
// After floyd if diagonal has <0 -> Neg cycle
void print(int a, int b) {
    int k = midMan[a][b];
    if (k == -1) v.PB(b);
    else {
        print(k, b);
        print(a, k);
    }
}
```

Tree Diameter:

```
// From any node 1st find the farthest node(f1) from that node
// Then from F1 the farthest node(f2)
// Tree diameter = dist[f1][f2];
```

Farthest node from a Given Node:

```
/* First find the tree diameter (f1,f2)
Then from given node dist[a][f1] & dist[a][f2] which one is maximum */
```

Topological Sort:

```
/* In degree 0 gula k queue te rakhbo then oi gula 1 ta kore queue theke ber kore tader adjancent node er indegree 1 minus korbo. If any in degree decreases to 0 then push the node in queue. jodi n ta node na hoi then topological sort nei */
```

Prufer Code to Tree:

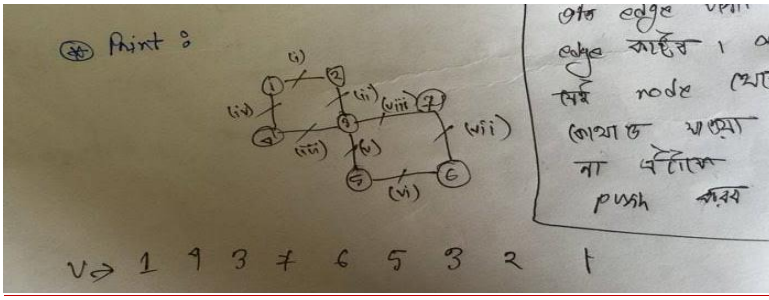
```
vector < int > prufer;
void pruferCodeToTree() {
    /*Stores number count of nodes in the prufer code*/
    map < int, int > mp;
    /* Set of integer absent in prufer code*/
    set < int > st;
    int len = prufer.size();
    int n = len + 2;
    /*Count frequency of nodes*/
    for (int i = 0; i < len; i++) {
        int t = prufer[i];
        mp[t]++;
    }
    /*Find the absent nodes*/
    for (int i = 1; i <= n; i++) {
        if (mp.find(i) == mp.end()) st.insert(i);
    }
    /*Connect Edges*/
    for (int i = 0; i < len; i++) {
        int a = prufer[i]; // First node
        /*Find the smallest number which is not present in prufer code now*/
        int b = * st.begin(); // Second node
        printf("%d %d¥n", a, b); // Edge of the tree
        st.erase(b); // Remove absent list
        mp[a]--; // Remove from prufer code
        if (mp[a] == 0) st.insert(a); // If a becomes absent

    }
    /*The final edge*/
    printf("%d %d¥n", * st.begin(), * st.rbegin());
}
// If the tree has n node then we can make n^(n-2) numbers of tree
```

Tree to Prufer Code:

```
/* Among all the leaves which one is minimum we will push that in the queue. And cut the edge from the tree. It will be continue until the length of prufer code is (n-2) */
```

Euler Circuit:

```
/* Euler Circuit: If we can visit every edge ONCE and can come back to the first node then this path is called Euler Circuit
Condition(Undirected):
(i) All degree even
(ii) graph connected
Condition(Directed):
(i) for each node InDegree = OutDegree
(ii) SCC
Print: Proti edge visit kore edge katbo( mark kore rakhbo). Pore jei node theke kothaou jawa jabena take v te push korbo.
*/
```



Euler Path:

```
/* Euler Path: If we can visit every edge ONCE and can go to the another node then this path is called Euler Path
Condition(Undirected):
(i) Except 2 every node has even degree
(ii) graph connected
(iii) Source & Destination have odd degree
Condition(Directed):
(i) for source (out - in) = +1
(ii) for destination (out - in) = -1
(iii) for others (out - in) = 0
(iv) after adding an edge from Destination to Source the graph will be SCC
Print: For undirected, graph traversal will be started from any odd degree node like euler circuit.
For directed, graph traversal will be strated from that node which has (out-in) = +1 like euler circuit*/
```

MST(KRUSKAL):

```
#define MAXN 100005
struct edge {
    int u, v, w;
    bool operator < (const edge & p) const {
        return w < p.w;
    }
};
int par[MAXN];
vector < edge > e;
int find(int r) {
    if (par[r] == r) return r;
    return par[r] = find(par[r]);
}
int mst(int n) {
    sort(e.begin(), e.end());
    for (int i = 1; i <= n; i++) par[i] = i;
    int count = 0, s = 0;
    for (int i = 0; i < (int) e.size(); i++) {
        int u = find(e[i].u);
        int v = find(e[i].v);
        if (u != v) {
            par[u] = v;
            count++;
            s += e[i].w;
            if (count == n - 1) break;
        }
    }
    ///check if a mst exist or not by count
    return s;
}
```

MST(PRIMS):

```
typedef vector < vector < pii > > Graph;
long long prim(Graph & g, vector < int > & pred) {
    int n = g.size();
    pred.assign(n, -1);
    vector < bool > vis(n);
    vector < int > prio(n, INT_MAX);
    prio[0] = 0;
```

```
        priority_queue < pii, vector < pii > , greater < pii > > q;
        q.push(make_pair(0, 0));
        long long res = 0;
        while (!q.empty()) {
            int d = q.top().first;
            int u = q.top().second;
            q.pop();
            if (vis[u])
                continue;
            vis[u] = true;
            res += d;
            for (int i = 0; i < (int) g[u].size(); i++) {
                int v = g[u][i].first;
                if (vis[v])
                    continue;
                int nprio = g[u][i].second;
                if (prio[v] > nprio) {
                    prio[v] = nprio;
                    pred[v] = u;
                    q.push(make_pair(nprio, v));
                }
            }
        }
        return res;
    }
    int main() {
        Graph g(3);
        g[0].push_back(make_pair(1, 10));
        vector < int > prio;
        long long res = prim(g, prio);
        cout << res << endl;
    }
```

MAXIMUM BIPARTITE MATCHING:

```
    /* যদি কোন গ্রাফ এর N টা নোড থাকে , এমন এই N টা নোডকে U , V দুইটা Independent  set এ বিভক্ত করা যাবে যাতে U Set  এর প্রত্যেকটা নোড এর সাথে V set  এর কোন না কোন নোড এর সাথে connect থাকবে । এই গ্রাফ এ কোন odd cycle থাকবে না। যেহেতু U ,V দুইটা মাত্রই set এ নোড গুলো বিভক্ত odd cycle থাকা possible ও না। আর Independent set U,V এর নিজেদের মধ্যে কোন কানেক শন থাকবে না। মানে U set এর কোন নোড নিজেদের মধ্যে connect ed থাকবে না।
     */
    int adj[MAX][MAX], deg[MAX], Left[MAX], Right[MAX], m, n;
    bool visited[MAX];
    bool bpm(int u) {
        for (int i = 0, v; i < deg[u]; i++) {
            v = adj[u][i];
            if (visited[v]) continue;
            visited[v] = true;
            if (Right[v] == -1 || bpm(Right[v])) {
                Right[v] = u, Left[u] = v;
                return true;
            }
        }
        return false;
    }
    int bipartiteMatching() { // Returns Maximum Matching
        memset(Left, -1, sizeof(Left));
        memset(Right, -1, sizeof(Right));
        int i, cnt = 0;
        for (i = 0; i < m; i++) {
            memset(visited, 0, sizeof(visited));
            if (bpm(i)) cnt++;
        }
        return cnt;
    }
```

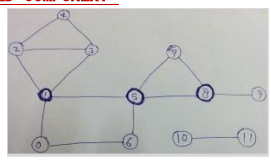
2-SAT:

```
    /* 1. The nodes need to be split. So change convert() accordingly.
    2. Using clauses, populate scc edges.
    3. Call possible, to find if a valid solution is possible or not.
    4. Dont forget to keep space for !A variables */
    struct SAT2 {
        SCC scc; /// This is from SCC Class
        SAT2(): bfscc(1) {}
        void clear() {
            scc.clear();
        }
        int convert(int n) { ///Change here. Depends on how input is provided
            int x = ABS(n);
            x--;
            x *= 2;
            if (n < 0) x ^= 1;
            return x;
        }
        void mustTrue(int a) { ///A is True
            scc.adj[a ^ 1].pb(a);
        }
        void orClause(int a, int b) { /// A || B clause
                //!a->b !b->a
                scc.adj[a ^ 1].pb(b);
                scc.adj[b ^ 1].pb(a);
            }
            /// Out of all possible option, only one is true
        void atMostOneClause(int a[], int n, int flag) {
                if (flag == 0) { /// At most one can be false
                    FOR(i, 0, n) {
                        a[i] = a[i] ^ 1;
                    }
                }
                FOR(i, 0, n) {
                    FOR(j, i + 1, n) {
                        orClause(a[i] ^ 1, a[j] ^ 1); /// !a || !b both being true not allowed
                    }
                }
            }
            ///Send n, total number of nodes, after expansion
        bool possible(int n) {
                scc.findSCC(n);

                FOR(i, 0, n) {
                        int a = i, b = i ^ 1;
                        ///Falls on same cycle a and !a.
                        if (scc.cycle[a] == scc.cycle[b]) return false;
                    }
                    ///Valid solution exists
                return true;
            }
            ///To determine if A can be true. It cannot be true, if a path exists from A to !A.
        int vis[SAT2NODE], qqq[SAT2NODE], bfscc;
        void bfs(int s) {
            bfscc++;
            int qs = 0, qt = 0;
            vis[s] = bfscc;
            qqq[qt++] = s;
            while (qs < qt) {
                s = qqq[qs++];
                FOR(i, 0, SZ(scc.adj[s]) - 1) {
                    int t = scc.adj[s][i];
                    if (vis[t] != bfscc) {
                        vis[t] = bfscc;
                        qqq[qt++] = t;
                    }
                }
            }
        }

    };
    sat2;
```

ERDOS AND GALLAI THEOREM:

```
// Given the degrees of the vertices of a graph, is it possible to construct such graph Input - the deg[] array
int deg[MM], n, degSum[MM], ind[MM], minVal[MM];
bool ErdosGallai() { // 1 indexed
    bool poss = true;
    int i, sum = 0, j, r;
    for (i = 1; i <= n; i++) {
        if (deg[i] >= n) poss = false;
        sum += deg[i];
    }
    //Summation of degrees has to be ODD and all degrees has to be < n - 1
    if (!poss || (sum & 1) || (n == 1 && deg[1] > 0)) return false;
    sort(deg + 1, deg + n + 1, greater < int > ());
    degSum[0] = 0;
    j = n;
    for (i = 1; i <= n; i++) {
        degSum[i] = degSum[i - 1] + deg[i]; //CONSTRUCTING: degSum
        for (; j >= 1 && deg[j] < i; j--
); //CONSTRUCTING: ind
        ind[i] = j + 1;
    }
    //CONSTRUCTING : minVal
    for (r = 1; r < n; r++) {
        j = ind[r];
        if (j == n + 1) minVal[r] = (n - r) * r;
        else if (j <= r) minVal[r] = degSum[n] - degSum[r];
        else {
            minVal[r] = degSum[n] - degSum[j - 1];
            minVal[r] += (j - r - 1) * r;
        }
    }
    //Checking : Erdos & Gallai Theorem
    for (r = 1; r < n; r++)
        if (degSum[r] > (r * (r - 1) + minVal[r])) return false;
    return true;
}
```

BICONNECTED COMPONENT:



```
/*In above graph, following are the biconnected components:
4-2 3-4 3-1 2-3 1-2
8-9
8-5 7-8 5-7
6-0 5-6 1-5 0-1
10-11
An undirected graph is called Biconnected if there are two vertex-disjoint paths between any two vertices.
Idea is to store visited edges in a stack while DFS on a graph and keep looking for Articulation Points (highlighted in above figure). As soon as an Articulation Point u is found, all edges visited while DFS from node u onwards will form one biconnected component. When DFS completes for one connected component, all edges present in stack will form a biconnected component.
If there is no Articulation Point in graph, then graph is biconnected and so there will be one biconnected component which is the graph itself.
*/
```

DYNAMIC PROGRAMMING:

LIS NlogK:

```
///input must be 0 indexed
vector < int > Sequence, I, L;
int LisNlogK() {
    int i;
    I.clear();
    L.clear();
    I.push_back(-INF);
    for (i = 1; i <= n; i++) I.push_back(INF);
    int LisLength = 0;
    for (i = 0; i < n; i++) {
        int low, high, mid;
        low = 0;
        high = LisLength;
        while (low <= high) {
            mid = (low + high) / 2;
            if (I[mid] < Sequence[i])
                low = mid + 1;
            else
                high = mid - 1;
        }
        I[low] = Sequence[i];
        if (LisLength < low)
            LisLength = low;
        L.push_back(low);
    }
    return LisLength;
}
void findSequence(int length) {
    int ind;
    for (int j = L.size() - 1; j >= 0; j--) {
        if (L[j] == length) {
            ind = j;
            break;
        }
    }
    stack < int > st;
    int mx = length - 1;
    st.push(Sequence[ind]);
    for (int i = ind - 1; i >= 0; i--) {
        if (L[i] == mx && Sequence[ind] > Sequence[i]) {
            st.push(Sequence[i]);
            ind = i;
            mx--;
        }
    }
    while (!st.empty()) {
        cout << st.top() << endl;
        st.pop();
    }

}
```

2D LIS (N log N):

```
typedef pair < int, int > pii;
pii p[100005];
set < pii > s[100005];
set < pii > ::iterator it, it1;
int main() {
    int n, i, lo, hi, mid, lb, k, t, cs = 1;
    scanf("%d", & n);
    for (i = 0; i < n; i++) scanf("%d %d", & p[i].first, & p[i].second);
    s[0].insert(p[0]);
    k = 0;
    for (i = 1; i < n; i++) {
        lo = 0;
        hi = k, lb = -1;
        while (lo <= hi) {
            mid = (lo + hi) / 2;
            it = s[mid].lower_bound(p[i]);
            if (it != s[mid].begin()) {
                it1 = it, it1--;
                if (( * it1).first == p[i].first) it--;
            }
            if (it != s[mid].begin() && ( * (--it)).second < p[i].second)
                lo = mid + 1, lb = max(lb, mid);
            else hi = mid - 1;
        }
        lb++;
```

```
            k = max(k, lb);
            it = s[lb].lower_bound(pii(p[i].first, -inf));
            if (it == s[lb].end() || (( * it).first > p[i].first
    || ( * it).second > p[i].second))
                s[lb].insert(p[i]);
            it = s[lb].upper_bound(p[i]);
            while (it != s[lb].end()) {
                if (( * it).first >= p[i].first && ( * it).second
    >= p[i].second) {
                    it1 = it, it1++;
                    s[lb].erase(it);
                    it = it1;
                } else break;
            }
        }
        printf("%d¥n", k + 1);
    }
```

LCS 1D:

```
    Outline: O(nm) algorithm foR the LCS With O(n) spAce
    int m[2][1000]; // instead of [1000][1000]
    for (i = M; i >= 0; i--) {
        ii = i & 1;
        for (j = N; j >= 0; j--) {
            if (i == M || j == N) {
                m[ii][j] = 0;
                continue;
            }
            if (s1[i] == s2[j]) m[ii][j] = 1 + m[1 - ii][j + 1];

            else m[ii][j] = max(m[ii][j + 1], m[1 - ii][j]);
        }
    }
    cout << m[0][0]; // if you want m[x][y], write m[x&1][y]
```

Matrix Chain Multiplication (MCM) O(MAX * MAX):

```
    #define MAX 100
    int row[MAX], col[MAX];
    int dp[MAX][MAX];
    bool visited[MAX][MAX];
    int f(int beg, int end) {
        if (beg >= end) return 0;
        if (visited[beg][end]) return dp[beg][end];
        int ans = 1 << 30; //২^৩০ কে ইনফিনিটি ধরছি
        for (int mid = beg; mid < end; mid++) //দুইভাগে ভাগ করছি

        {
            int opr_left = f(beg, mid); //opr = multiplication op
    eration
            int opr_right = f(mid + 1, end);
            int opr_to_multiply_left_and_right = row[beg] * col[m
    id] * col[end];
            int total = opr_left + opr_right + opr_to_multiply_le
    ft_and_right;
            ans = min(ans, total);
        }
        visited[beg][end] = 1;
        dp[beg][end] = ans;
        return dp[beg][end];
    }
    int main() {
        int n; cin >> n;
        FOR(i,0, n-1) cin >> row[i] >> col[i];
        cout << f(0, n - 1) << endl;
    }
```

HISTOGRAM:

```
    #define SIZE 120
    int arr[SIZE][SIZE];
    int cum[SIZE][SIZE];
    int histogram(int row, int col) {
        stack < pii > st;
        st.push(pii(-INF, 0));
        int lft[SIZE], rght[SIZE];
        rep(i, col) {
            while (st.top().ff >= cum[row][i]) st.pop();
            lft[i] = st.top().ss;
            st.push(pii(cum[row][i], i));
        }
        while (!st.empty()) st.pop();
        st.push(pii(-INF, col + 1));
        ROF(i, col, 1) {

            while (st.top().ff >= cum[row][i]) st.pop();
            rght[i] = st.top().ss;
            st.push(pii(cum[row][i], i));
        }
        int res = -INF;
        rep(i, col) {
            res = max(res, cum[row][i] * (rght[i] - lft[i] - 1));

        }
        return res;
    }
```

LONGEST PALINDROME (MANACHER ALGORITHM):

```
    // Transform S into T.
    // For example, S = "abba", T = "^#a#b#b#a#$".
    // ^ and $ signs are sentinels appended to each end to avoid
    bounds checking
    string preProcess(string s) {
        int n = s.length();
        if (n == 0) return "^$";
        string ret = "^";
        for (int i = 0; i < n; i++)
            ret += "#" + s.substr(i, 1);

        ret += "#$";
        return ret;
    }
    string longestPalindrome(string s) {
        string T = preProcess(s);
        int n = T.length();
        int * P = new int[n];
        int C = 0, R = 0;
        for (int i = 1; i < n - 1; i++) {
            int i_mirror = 2 * C - i; // equals to i' = C - (i-
    C)

            P[i] = (R > i) ? min(R - i, P[i_mirror]) : 0;

            // Attempt to expand palindrome centered at i
            while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
                P[i]++;

            // If palindrome centered at i expand past R,
            // adjust center based on expanded palindrome.
            if (i + P[i] > R) {
                C = i;
                R = i + P[i];
            }
        }
        // Find the maximum element in P.
        int maxLen = 0;
        int centerIndex = 0;
        for (int i = 1; i < n - 1; i++) {
            if (P[i] > maxLen) {
                maxLen = P[i];
                centerIndex = i;
            }
        }
        delete[] P;
        return s.substr((centerIndex - 1 - maxLen) / 2, maxLen);

    }
```

2d MAX SUM: (dipta007)

```
    struct kadane //Structure for 1D-Kadane Algorithm
    {
        int u, d;
        ll sum, area;
```

```
    kadane() {
        u = -1, d = -1, sum = 0, area = 0;
    }
};
ll a[104][104], jaBerHoilo;
pii twoPoint(ll kad[104], int n, ll k) {
    int low = 0, maxL = -1, maxR = -1, maxLen = 0;
    ll maxSum = 0, sum = 0;
    FOR(high, 0, n - 1) {
        sum += kad[high];
        while (sum > k) {
            sum -= kad[low];
            low++;
        }
        int len = high - low + 1;
        if (len > maxLen) {
            maxSum = sum; maxLen = len;
            maxL = low; maxR = high;
        } else if (len == maxLen && sum < maxSum) {
            maxSum = sum; maxLen = len;
            maxL = low; maxR = high;
        }
    }
    jaBerHoilo = maxSum;
    return pii(maxL, maxR);
}
int main() {
    int t;
    getI(t);
    FOR(ci, 1, t) {
        int n, m, k;
        getIII(n, m, k);
        FOR(i, 0, n - 1)
            FOR(j, 0, m - 1)
                getL(a[i][j]);
        int r = n, c = m, maxL, maxR, maxU, maxD;
        ll maxSum = 0, maxArea = 0, kad[r];
        for (int left = 0; left < c; left++) {
            CLR(kad);
            for (int right = left; right < c; right++) {
                for (int i = 0; i < r; i++) {
                    kad[i] = kad[i] + a[i][right];
                }
                pii res = twoPoint(kad, r, k);
                if (res.ff == -1 || res.ss == -1) continue;
                kadane maxy;
                maxy.sum = jaBerHoilo;
                maxy.u = res.ff; maxy.d = res.ss;
                maxy.area = (ll) abs(maxy.d - maxy.u + 1) * (
ll) abs(right - left + 1);
                if (maxy.area > maxArea) {
                    maxArea = maxy.area;
                    maxSum = maxy.sum;
                    maxL = left; maxR = right;
                    maxU = maxy.u; maxD = maxy.d;
                } else if (maxy.area == maxArea && maxy.sum <
maxSum) {
                    maxArea = maxy.area;
                    maxSum = maxy.sum;
                    maxL = left; maxR = right;
                    maxU = maxy.u; maxD = maxy.d;
                }
            }
        } ///end of 2d kadane
        printf("Case #%d: %lld %lld¥n", ci, maxArea, maxSum);
    }
}
```

2d MAX SUM: (mamun4122)

```
#define SIZE 105
int arr[SIZE][SIZE], vertical[SIZE][SIZE], cum[SIZE][SIZE];
int tmp[SIZE];
int twopointer(int frst, int scnd) {
    CLR(tmp);
    rep(i, m) tmp[i]=vertical[scnd][i]-vertical[frst-1][i];
    int lft = 1, rght = 1, res = -INF, sum = 0;
    while (rght <= m) {
        if (sum + tmp[rght] >= tmp[rght]) sum += tmp[rght];
        else {
            while(lft< rght && sum + tmp[rght] < tmp[rght]) {
                sum -= tmp[lft];
                lft++;
            }
            sum += tmp[rght];
        }
        res = max(res, sum);
        rght++;
    }
    res = max(res, sum);
    return res;
}
int findmaximumsum(int r, int c) {
    int n = r, m = c;
    CLR(vertical);
    CLR(cum);
    ///first find the vertical cum sum of every row
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++)
            vertical[i][j] = vertical[i - 1][j] + arr[i][j];

    }
    int ans = -INT_MAX;
    for (int first = 1; first <= n; first++) {
        for (int scnd = first; scnd <= n; scnd++) {
            ans = max(ans, twopointer(first, scnd));
        }
    }
    return ans;
}
```

COIN CHANGE (II):

```
/*In a strange shop there are n types of coins of value A1, A
2 ... An. You have to find the number of ways you can make M
using the coins. You can use any coin at most M times.*/
int t, n, m, val[105];
int dp[105][10005];
#define mod 100000007
int main() {
    getI(t);
    rep(cs, t) {
        getII(n, m);
        int x;
        rep(i, n) getI(val[i]);
        rep(i, n) dp[i][0] = 1;
        rep(i, m) dp[0][i] = 0;
        rep(i, n) {
            rep(j, m) {
                if (j < val[i]) dp[i][j] = dp[i - 1][j] % mod
;
                else dp[i][j] = ((dp[i - 1][j] % mod + dp[i][
j - val[i]] % mod) % mod);
            }
        }
        printf("Case %d: %d¥n", cs, dp[n][m]);
    }
}
```

COIN CHANGE (III):

```
/*In a strange shop there are n types of coins of value A1, A
2 ... An. C1, C2, ... Cn denote the number of coins of value
A1, A2 ... An respectively. You have to find the number of di
fferent values (from 1 to m), which can be produced using the
se coins.*/
int t, n, m;
int arr[105], val[105];
int dp[100005], need[100005];
int main() {
    getI(t);
    rep(cs, t) {
```

```
            CLR(dp);
            getII(n, m);
            rep(i, n) getI(val[i]);
            rep(i, n) getI(arr[i]);
            dp[0] = 1;
            int ans = 0;
            rep(i, n) {
                CLR(need);
                for (int j = val[i]; j <= m; j++) {
                    if (!dp[j] && dp[j - val[i]] && need[j - val[i]] + 1 <= arr[i]) {
                        ans++;
                        dp[j] = 1;
                        need[j] = need[j - val[i]] + 1;
                    }
                }
            }
            printf("Case %d: %d\n", cs, ans);
        }
    }
```

DIGIT DP (diptaO07) :

```
const int NX = 70;
ll dp[2][2][NX][NX];
int vis[2][2][NX][NX];
int lim, tt;
vector < int > inp;
ll DP(int pos, int isSmall, int isStart, int value) {
    if (pos == lim) return value;
    ll & ret = dp[isSmall][isStart][pos][value];
    int & v = vis[isSmall][isStart][pos][value];
    if (v == tt) return ret;
    v = tt;
    int ses = isSmall ? 9 : inp[pos];
    int i;
    ret = 0;
    if (!isStart) {
        for (i = 0; i <= ses; i++) {
            ret += DP(pos + 1, isSmall | i < inp[pos], 0, (i == 0) + value);
        }
    } else {
        for (i = 1; i <= ses; i++) {
            ret += DP(pos + 1, isSmall | i < inp[pos], 0, (i == 0) + value);
        }
        ret += DP(pos + 1, 1, 1, 0);
    }
    return ret;
}
ll Cal(ll x) {
    if (x < 0) return 0;
    if (x <= 9) return 1;
    inp.clear();
    while (x) {
        inp.pb(x % 10);
        x /= 10;
    }
    reverse(inp.begin(), inp.end());
    lim = inp.size();
    tt++;
    return DP(0, 0, 1, 0) + 1; //for '0' case  (+1)
}
int main() {
    int cs, t;
    getI(t);
    for (cs = 1; cs <= t; cs++) {
        ll n, m;
        getLL(n, m);
        ll ans = Cal(m) - Cal(n - 1);
        printf("Case %d: %lld\n", cs, ans);
    }
}
```

DIGIT DP (mamun4122) :

```
int tot;
vector < int > dig;
ll dp[20][2][200][2];
ll call(int pos, int flag, int sum, int strt) {
    if (pos == tot) return sum;
    ll & ret = dp[pos][flag][sum][strt];
    if (ret != -1) return ret;
    ll ans = 0;
    if (pos == 0) {
        repI(i, dig[pos] + 1) {
            ans += call(pos + 1, i == dig[pos], sum + (i == 0 && strt), (strt || i != 0));
        }
    } else {
        if (flag) {
            for (int i = 0; i <= dig[pos]; i++) {
                ans += call(pos + 1, i == dig[pos], sum + (i == 0 && strt), (strt || i != 0));
            }
        } else {
            repI(i, 10)
            ans += call(pos + 1, 0, sum + (i == 0 && strt), (strt || i != 0));
        }
    }
    return ret = ans;
}
void calc(ll num) {
    dig.clear();
    while (num) {
        dig.push_back(num % 10);
        num /= 10;
    }
    reverse(ALL(dig));
    tot = dig.size();
}
int main() {
    int t, n, m;
    getI(t);
    rep(cs, t) {
        SET(dp);
        ll a, b;
        getLL(a, b);
        a--;
        ll ansa, ansb;
        if (a < 0) ansa = 0;
        else if (a < 10) ansa = 1;
        else {
            calc(a);
            ansa = call(0, 0, 0, 0) + 1;
        }
        SET(dp);
        calc(b);
        ansb = call(0, 0, 0, 0) + 1;
        printf("Case %d: %lld\n", cs, ansb - ansa);
    }
}
```

EDIT DISTANCE:

```
int MOD = 1000000007;
int dp[4000][4000];
int main() {
    int t;
    getI(t);
    getchar();
    for (int ci = 1; ci <= t; ci++) {
        string a, b;
        cin >> a >> b;
        int la = a.size();
        int lb = b.size();
        for (int i = 0; i <= la; i++)
            dp[0][i] = i;
        for (int i = 0; i <= lb; i++)
            dp[i][0] = i;
```

```
            for (int i = 1; i <= lb; i++) {
                for (int j = 1; j <= la; j++) {
                    if (a[j - 1] == b[i - 1]) {
                        dp[i][j] = dp[i - 1][j - 1];
                    } else {
                        dp[i][j] = min(dp[i - 1][j], min(dp[i - 1
    ][j - 1], dp[i][j - 1])) + 1;
                    }
                }
            }
            printf("%d¥n", dp[lb][la]);
        }
    }
```

GAME THEORY:

NIM:

```
/*নিম-
গেম এ দুইজন খেলোয়ার আর কিছু পাথরের স্তুপ(pile) থাকে। প্রতি চালে একজন খেলোয়াড় যেকোনো একটা স্তুপ থেকে এক বা একাধিক পাথর তুলে নিতে পারে। কেও চাল দিতে ব্যার্থ হলে হেরে যাবে। অর্থাৎ শেষ পাথরটা যে তুলে নিয়েছে সে গেমে জিতবে।
 */
if (xorsum > 0) first win
else second win
```

Spurge Grundy:

```
int grundy[600][600];
int dirx[]={-2,-3,-2,-1,-1,1};
int diry[]={1,-1,-1,-2,-3,-2};
int calc(int x, int y) {
    if (grundy[x][y] != -1)
        return grundy[x][y];
    set < int > st;
    for (int i = 0; i < 6; i++) {
        int posx = x + dirx[i];
        int posy = y + diry[i];
        if (posx >= 0 && posy >= 0)
            st.insert(calc(posx, posy));
    }
    int ans = 0;
    while (st.contains(ans)) ans++;
    return grundy[x][y] = ans;
}
int main() {
    int i, j, t, cs, n;
    getI(t);
    SET(grundy);
    rep(cs, t) {
        int ans = 0;
        printf("Case %d: ", cs);
        getI(n);
        rep(i, n) {
            int x, y;
            getII(x, y);
            ans ^= calc(x, y);
        }
        if (ans) puts("Alice");
        else puts("Bob");
    }
}
```

MINMAX :

```
const int MAXN = 100005;
int dp[MAXN];
bool vis[MAXN];
int moves[]={1,3,5};
bool valid_move(int x) {
    return x >= 0;
}
bool MINMAX(int x) {
    if (x == 0) return false; ///LOSE
    if (vis[x]) return dp[x];
    vis[x] = 1;
    FOR(i, 0, 2) {
        if (valid_move(x - moves[i]) && !MINMAX(x - moves[i])
)
            return dp[x] = true;
    }
    return dp[x] = false;
}
int main() {
    int n;
    getI(n);
    CLR(vis);
    if (MINMAX(n)) printf("First");
    else printf("Second");
}
```

DATA STRUCTURE:

UNION FIND/DISJOINT SET:

```
class UnionFind { // OOP style
    private:
        vi p, rank, setSize; // remember: vi is vector<int>
    int numSets;
    public:
        UnionFind(int N) {
            setSize.assign(N, 1);
            numSets = N;
            rank.assign(N, 0);
            p.assign(N, 0);
            for (int i = 0; i < N; i++) p[i] = i;
        }
    int findSet(int i) {
        return (p[i] == i) ? i : (p[i] = findSet(p[i]));
    }
    bool isSameSet(int i, int j) {
        return findSet(i) == findSet(j);
    }
    void unionSet(int i, int j) {
        if (!isSameSet(i, j)) {
            numSets--;
            int x = findSet(i), y = findSet(j);
            // rank is used to keep the tree short
            if (rank[x] > rank[y]) {
                p[y] = x;
                setSize[x] += setSize[y];
            } else {
                p[x] = y;
                setSize[y] += setSize[x];
                if (rank[x] == rank[y]) rank[y]++;
            }
        }
    }
    int numDisjointSets() {
        return numSets;
    }
    int sizeOfSet(int i) {
        return setSize[findSet(i)];
    }
};
```

UNION FIND (MAMUN4122) :

```
int find_representative(int r) {
    if (par[r] == r) return r;
    else {
        return par[r] = find_representative(par[r]);
    }
}
```

SEGMENT TREE:

```
#define mx 100001
int arr[mx];
int tree[mx * 3];
void init(int node, int b, int e) {
    if (b == e) {
        tree[node] = arr[b];
        return;
    }
    init(Left, b, mid);
    init(Right, mid + 1, e);
    tree[node] = tree[Left] + tree[Right];
}
int query(int node, int b, int e, int i, int j) {
    if (i > e || j < b) return 0;
    if (b >= i && e <= j) return tree[node];
```

```
        int p1 = query(Left, b, mid, i, j);
        int p2 = query(Right, mid + 1, e, i, j);
        return p1 + p2;
    }
    void update(int node, int b, int e, int i, int newvalue) {
        if (i > e || i < b) return;
        if (b >= i && e <= i) {
            tree[node] = newvalue;
            return;
        }
        update(Left, b, mid, i, newvalue);
        update(Right, mid + 1, e, i, newvalue);
        tree[node] = tree[Left] + tree[Right];
    }
    ///lazy with propagation
    void Propagate(int at, int L, int R) {
        int mid = (L + R) / 2;
        int left_at = at * 2, left_L = L, left_R = mid;
        int right_at = at * 2 + 1, right_L = mid + 1, right_R = R
    ;
        toggle[at] = 0;
        toggle[left_at] ^= 1;
        toggle[right_at] ^= 1;
        on[left_at] = left_R - left_L + 1 - on[left_at];
        on[right_at] = right_R - right_L + 1 - on[right_at];
    }
    void update(int at, int L, int R, int l, int r) {
        if (r < L || R < l) return;
        if (l <= L && R <= r) {
            toggle[at] ^= 1;
            on[at] = R - L + 1 - on;
            return;
        }
        if (toggle[at]) Propagate(at, L, R);
        int mid = (L + R) / 2;
        update(at * 2, L, mid, l, r);
        update(at * 2 + 1, mid + 1, R, l, r);
        on[at] = on[at * 2] + on[at * 2 + 1];
    }
    int query(int at, int L, int R, int l, int r) {
        if (r < L || R < l) return;
        if (l <= L && R <= r) return on[at];
        if (toggle[at]) Propagate(at, L, R);
        int mid = (L + R) / 2;
        int x = query(at * 2, L, mid, l, r);
        int y = query(at * 2 + 1, mid + 1, R, l, r);
        return x + y;
    }
```

Binary Indexed Tree (BIT):

```
    int query(int idx) {
        int sum = 0;
        while (idx > 0) {
            sum += tree[idx];
            idx -= idx & (-idx);
        }
        return sum;
    }
    void update(int idx, int x, int n)
    //n is the size of the array, x is the number to add
    {
        while (idx <= n) {
            tree[idx] += x;
            idx += idx & (-idx);
        }
    }
```

Range Minimum Query (RMQ) :

```
    const int inf = (1 << 28);
    template < typename t > t MIN3(t a, t b, t c) {
        return min(a, min(b, c));
    }
    const int sz = 100005;
    int BLOCK[400];
    int arr[sz];
    int getId(int indx, int blockSZ) {
        return indx / blockSZ;
    }
    void init(int sz) {
        for (int i = 0; i <= sz; i++) BLOCK[i] = inf;
    }
    void update(int val, int indx, int blockSZ) {
        int id = getId(indx, blockSZ);
        BLOCK[id] = min(BLOCK[id], val);
    }
    int query(int L, int R, int blockSZ) {
        int lid = getId(L, blockSZ);
        int rid = getId(R, blockSZ);
        if (lid == rid) {
            int ret = inf;
            for (int i = L; i <= R; i++) ret = min(ret, arr[i]);

            return ret;
        }
        int m1 = inf, m2 = inf, m3 = inf;
        for (int i = L; i < (lid + 1) * blockSZ; i++) m1 = min(m1
    , arr[i]);
        for (int i = lid + 1; i < rid; i++) m2 = min(m2, BLOCK[i]
    );
        for (int i = rid * blockSZ; i <= R; i++) m3 = min(m3, arr
    [i]);
        return MIN3(m1, m2, m3);
    }
    int main() {
        int N, Q;
        scanf("%d %d", & N, & Q);
        int blockSZ = sqrt(N);
        init(blockSZ);
        for (int i = 0; i < N; i++) {
            int x;
            scanf("%d", & x);
            arr[i] = x;
            update(x, i, blockSZ);
        }
        while (Q--) {
            int x, y;
            scanf("%d %d", & x, & y);
            printf("%d¥n", query(x, y, blockSZ));
        }
    }
    //
    //getId ফাংশনের কাজ হল কোন index কত নাম্বার block এ তা বে
    র করে দেয়া।
    //init ফাংশন সবগুলা block কে infinity ভ্যালু নিয়ে initialize
    করে নিচ্ছে
    //update ফাংশন দিয়ে কোন একটা নির্দিষ্ট index এর ভ্যালু আপডেট
    করে দেয়া হচ্ছে।
    //query ফাংশন দিয়ে x to y রেঞ্জের result calculation করা হচ্ছ
    ে।
    //Line 25 এ যদি রেঞ্জ পুরোটা কোন একটা নির্দিষ্ট block এর sub p
    art হয়ে থাকে, তাহলে main Array থেকে result calculation করে
    দিবে।
    //Line 32, যদি রেঞ্জ এর lower bound এর কিছু অংশ নির্দিষ্ট একটা
    block এর sub part হয়ে থাকে তাহলে শুধু সেইটুক sub part এর re
    sult main Array থেকে calculate করে দিবে।
    //Line 34, যদি রেঞ্জ এর upper bound এর কিছু অংশ নির্দিষ্ট একটা
    block এর sub part হয়ে থাকে তাহলে শুধু সেইটুক sub part এর re
    sult main Array থেকে নিবে।
```

PREFIX TRIE:

```
    #define mx 26
    struct node {
        bool endmark;
        node * next[mx + 1];
        node() {
            endmark = 0;
            for (int i = 0; i < mx; i++)
                next[i] = NULL;
        }
```

```
} * root;
void insert(char * str, int len) {
    node * curr = root;
    for (int i = 0; i < len; i++) {
        int id = str[i] - 'a';
        if (curr - > next[id] == NULL)
            curr - > next[id] = new node();
        curr = curr - > next[id];
    }
    curr - > endmark = 1;
}
bool search(char * str, int len) {
    node * curr = root;
    for (int i = 0; i < len; i++) {
        int id = str[i] - 'a';
        if (curr - > next[id] == NULL) return false;
        curr = curr - > next[id];
    }
    return curr - > endmark; /// returns 1 or 0
}
void del(node * cur) /// send root here
    {
        for (int i = 0; i < mx; i++)
            if (cur - > next[i])
                del(cur - > next[i]);
        delete(cur);
    }
int main() {
    root = new node();
    int num_word;
    cin >> num_word;
    for (int i = 1; i <= num_word; i++) {
        char str[50];
        scanf("%s", str);
        insert(str, strlen(str));
    }
    int query;
    cin >> query;
    for (int i = 1; i <= query; i++) {
        char str[50];
        scanf("%s", str);
        if (search(str, strlen(str))) puts("FOUND");
        else puts("NOT FOUND");
    }
    del(root);
}
```

STRING ALGORITHMS:

KMP:

```
void computeLPSArray(char * pat, int M, int * lps);
void KMPSearch(char * pat, char * txt) {
    int M = strlen(pat);
    int N = strlen(txt);
    int * lps = (int * ) malloc(sizeof(int) * M);
    int j = 0;
    computeLPSArray(pat, M, lps);
    int i = 0; // index for txt[]
    while (i < N) {
        if (pat[j] == txt[i]) {
            j++;
            i++;
        }
        if (j == M) {
            printf("Found pattern at index %d ¥n", i - j);
            j = lps[j - 1];
        } else if (pat[j] != txt[i]) {
            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;
        }
    }
    free(lps);
}
void computeLPSArray(char * pat, int M, int * lps) {
    int len = 0;
    int i;
    lps[0] = 0;
    i = 1;
    // the loop calculates lps[i] for i = 1 to M-1
    while (i < M) {
        if (pat[i] == pat[len]) {
            len++;
            lps[i] = len;
            i++;
        } else // (pat[i] != pat[len])
        {
            if (len != 0) {
                // This is tricky. Consider the example AAACA
AAA and i = 7.
                len = lps[len - 1];
            } else {
                lps[i] = 0;
                i++;
            }
        }
    }
}
int main() {
    char * txt = "ABABDABACDABABCABAB";
    char * pat = "ABABCABAB";
    KMPSearch(pat, txt);
}
```

Z ALGORITHM:

```
const int NX = 1e5 + 10; // string size
char text[NX];
int Z[NX];
void Z_Algorithm() {
        int position, starting_point, ending_point;
        int sz = strlen(text);
        Z[0] = sz; // always ;
        for (position = 1, starting_point = 0, ending_point =
0; position < sz; position++) {
            if (position <= ending_point) Z[position] = min(e
nding_point - position + 1, Z[position - starting_point]);
            while (position + Z[position] < sz && text[Z[posi
tion]] == text[position + Z[position]]) ++Z[position];
            if (position + Z[position] - 1 > ending_point) //
need to update
                starting_point = position, ending_point = pos
ition + Z[position] - 1;
        }
    }
    /******prefix==suffix**********/
for (i = sz - 1; i >= 0; i--) {
    if (Z[i] == sz - i) // suffix matches
}
/**********************/
bool zAlgorithm(string pattern, string target) {
    string s = pattern + '$' + target;
    int n = s.length();
    vector < int > z(n, 0);

    int goal = pattern.length();
    int r = 0, l = 0, i;
    for (int k = 1; k < n; k++) {
        if (k > r) {
            for (i = k; i < n && s[i] == s[i - k]; i++);
            if (i > k) {
                z[k] = i - k;
                l = k;
                r = i - 1;
            }
        } else {
            int kt = k - l, b = r - k + 1;
```

```
            if (z[kt] > b) {
                for (i = r + 1; i < n && s[i] == s[i - k]; i++);
                z[k] = i - k;
                l = k;
                r = i - 1;
            }
        }
        if (z[k] == goal)
            return true;
    }
    return false;
}
```

AHO-CORASICK:

```
#define MX 100 //small string length
int m, n, res;
typedef pair < int, int > Point;
struct NODE {
    int cnt;
    bool vis;
    NODE * next[27];
    vector < NODE * > out;
    NODE() {
        for (int i = 0; i < 27; i++) {
            next[i] = NULL;
        }
        out.clear();
        vis = false;
        cnt = 0;
    }~NODE() {
        for (int i = 1; i < 27; i++)
            if (next[i] != NULL && next[i] != this)
                delete next[i];
    }
} * root;
void buildtrie(char dictionary[][MX], int n) // processing the dictionarytionary
{
    root = new NODE();
    /*usual trie part*/
    for (int i = 0; i < n; i++) {
        NODE * p = root;
        for (int j = 0; dictionary[i][j]; j++) {
            char c = dictionary[i][j] - 'a' + 1;
            if (!p - > next[c])
                p - > next[c] = new NODE();
            p = p - > next[c];
        }
    }
    /* Pushing the nodes adjacent to root into queue */
    queue < NODE * > q;
    for (int i = 0; i < 27; i++) {
        if (!root - > next[i])
            root - > next[i] = root;
        else {
            q.push(root - > next[i]);
            root - > next[i] - > next[0] = root; // ->next[0] = back Pointer
        }
    }
    /* Building Aho-Corasick tree */
    while (!q.empty()) {
        NODE * u = q.front(); //parent node
        q.pop();
        for (int i = 1; i < 27; i++) {
            if (u - > next[i]) {
                NODE * v = u - > next[i]; // child node
                NODE * w = u - > next[0]; // back pointer of parent node
                while (!w - > next[i]) // Until the char(i+'a'-1) child is found
                    w = w - > next[0]; // go up and up to back pointer.
                v - > next[0] = w = w - > next[i]; // back pointer of v will be found child above.
                w - > out.push_back(v); // out will be used in dfs step.
                // here w is the new found match node.
                q.push(v); // Push v into queue.
            }
        }
    }
}
void aho_corasick(NODE * p, char * word) // Third step, processing the text.
{
    for (int i = 0; word[i]; i++) {
        char c = word[i] - 'a' + 1;
        while (!p - > next[c])
            p = p - > next[0];
        p = p - > next[c];
        p - > cnt++;
    }
}
int dfs(NODE * p) // DFS for counting.
{
    if (p - > vis) return p - > cnt;
    for (int i = 0; i < p - > out.size(); i++)
        p - > cnt += dfs(p - > out[i]);
    p - > vis = true;
    return p - > cnt;
}
char query[1000100];
char dictionary[MX][MX];
int main() {
    int t, tc, y, z;
    int i, j, k, l, h;
    char ch;
    scanf("%d", & tc);
    for (t = 1; t <= tc; t++) {
        int n;
        scanf("%d", & n);
        scanf("%s", query);
        for (int i = 0; i < n; ++i) {
            scanf("%s", dictionary[i]);
        }
        buildtrie(dictionary, n);
        aho_corasick(root, query);
        printf("Case %d:¥n", t);
        for (int i = 0; i < n; i++) {
            NODE * p = root;
            for (int j = 0; dictionary[i][j]; j++) {
                char c = dictionary[i][j] - 'a' + 1;
                p = p - > next[c];
            }
            printf("%d¥n", dfs(p));
        }
        delete root;
    }
}
```

SUFFIX ARRAY:

```
#define MAX_N 100010 // second approach: O(n log n)
char T[MAX_N]; // the input string, up to 100K characters
int n; // the length of input string
int RA[MAX_N], tempRA[MAX_N]; // rank array and temporary rank array
int SA[MAX_N], tempSA[MAX_N]; // suffix array and temporary suffix array
int c[MAX_N]; // for counting/radix sort
char P[MAX_N]; // the pattern string (for string matching)
int m; // the length of pattern string
int Phi[MAX_N]; // for computing longest common prefix
int PLCP[MAX_N];
int LCP[MAX_N]; // LCP[i] stores the LCP between previous suffix T+SA[i-1]
// and current suffix T+SA[i]
```

```
bool cmp(int a, int b) {
        return strcmp(T + a, T + b) < 0;
    } // compare
void constructSA_slow() { // cannot go beyond 1000 characters
    for (int i = 0; i < n; i++) SA[i] = i; // initial SA: {0, 1, 2, ..., n-1}
    sort(SA, SA + n, cmp); // sort: O(n log n) * compare: O(n) = O(n^2 log n)
}
void countingSort(int k) { // O(n)
    int i, sum, maxi = max(300, n); // up to 255 ASCII chars or length of n
    memset(c, 0, sizeof c); // clear frequency table
    for (i = 0; i < n; i++) // count the frequency of each integer rank
        c[i + k < n ? RA[i + k] : 0]++;
    for (i = sum = 0; i < maxi; i++) {
        int t = c[i];
        c[i] = sum;
        sum += t;
    }
    for (i = 0; i < n; i++) // shuffle the suffix array if necessary
        tempSA[c[SA[i] + k < n ? RA[SA[i] + k] : 0]++] = SA[i];
    for (i = 0; i < n; i++) // update the suffix array SA
        SA[i] = tempSA[i];
}
void constructSA() { // this version can go up to 100000 characters
    int i, k, r;
    for (i = 0; i < n; i++) RA[i] = T[i]; // initial rankings
    for (i = 0; i < n; i++) SA[i] = i; // initial SA: {0, 1, 2, ..., n-1}
    for (k = 1; k < n; k <<= 1) { // repeat sorting process log n times
        countingSort(k); // actually radix sort: sort based on the second item
        countingSort(0); // then (stable) sort based on the first item
        tempRA[SA[0]] = r = 0; // re-ranking; start from rank r = 0
        for (i = 1; i < n; i++) // compare adjacent suffixes
            tempRA[SA[i]] = // if same pair => same rank r; otherwise, increase r
             (RA[SA[i]] == RA[SA[i - 1]] && RA[SA[i] + k] == RA[SA[i - 1] + k]) ? r : ++r;
        for (i = 0; i < n; i++) // update the rank array RA
            RA[i] = tempRA[i];
        if (RA[SA[n - 1]] == n - 1) break; // nice optimization trick
    }
}
void computeLCP_slow() {
    LCP[0] = 0; // default value
    for (int i = 1; i < n; i++) { // compute LCP by definition
        int L = 0; // always reset L to 0
        while (T[SA[i] + L] == T[SA[i - 1] + L]) L++; // same L-th char, L++
        LCP[i] = L;
    }
}
void computeLCP() {
    int i, L;
    Phi[SA[0]] = -1; // default value
    for (i = 1; i < n; i++) // compute Phi in O(n)
        Phi[SA[i]] = SA[i - 1]; // remember which suffix is behind this suffix
    for (i = L = 0; i < n; i++) { // compute Permuted LCP in O(n)
        if (Phi[i] == -1) {
            PLCP[i] = 0;
            continue;
        } // special case
        while (T[i + L] == T[Phi[i] + L]) L++; // L increased max n times
        PLCP[i] = L;
        L = max(L - 1, 0); // L decreased max n times
    }
    for (i = 0; i < n; i++) // compute LCP in O(n)
        LCP[i] = PLCP[SA[i]]; // put the permuted LCP to the correct position
}
pii stringMatching() { // string matching in O(m log n)
        int lo = 0, hi = n - 1, mid = lo; // valid matching = [0..n-1]
        while (lo < hi) { // find lower bound
            mid = (lo + hi) / 2; // this is round down
            int res = strncmp(T + SA[mid], P, m); // try to find P in suffix 'mid'
            if (res >= 0) hi = mid; // prune upper half (notice the >= sign)
            else lo = mid + 1; // prune lower half including mid
        } // observe `=' in "res >= 0" above
        if (strncmp(T + SA[lo], P, m) != 0) return pii(-1, -1); // if not found
        pii ans;
        ans.first = lo;
        lo = 0;
        hi = n - 1;
        mid = lo;
        while (lo < hi) { // if lower bound is found, find upper bound
            mid = (lo + hi) / 2;
            int res = strncmp(T + SA[mid], P, m);
            if (res > 0) hi = mid; // prune upper half
            else lo = mid + 1; // prune lower half including mid
        } // (notice the selected branch when res == 0)
        if (strncmp(T + SA[hi], P, m) != 0) hi--; // special case
        ans.second = hi;
        return ans;
    } // return lower/upperbound as first/second item of the pair, respectively
pii LRS() { // returns a pair (the LRS length and its index)
    int i, idx = 0, maxLCP = -1;
    for (i = 1; i < n; i++) // O(n), start from i = 1
        if (LCP[i] > maxLCP)
            maxLCP = LCP[i], idx = i;
    return pii(maxLCP, idx);
}
int owner(int idx) {
    return (idx < n - m - 1) ? 1 : 2;
}
pii LCS() { // returns a pair (the LCS length and its index)
    int i, idx = 0, maxLCP = -1;
    for (i = 1; i < n; i++) // O(n), start from i = 1
        if (owner(SA[i]) != owner(SA[i - 1]) && LCP[i] > maxLCP)
            maxLCP = LCP[i], idx = i;
    return pii(maxLCP, idx);
}
int main() {
    //printf("Enter a string T below, we will compute its Suffix Array:¥n");
    strcpy(T, "GATAGACA");
    n = (int) strlen(T);
```

```
    T[n++] = '$';
    // if '¥n' is read, uncomment the next line
    //T[n-1] = '$'; T[n] = 0;
    constructSA_slow(); // O(n^2 log n)
    printf("The Suffix Array of string T = '%s' is shown belo
w (O(n^2 log n) version):¥n", T);
    printf("i¥tSA[i]¥tSuffix¥n");
    for (int i = 0; i < n; i++) printf("%2d¥t%2d¥t%s¥n", i, S
A[i], T + SA[i]);
    constructSA(); // O(n log n)
    printf("¥nThe Suffix Array of string T = '%s' is shown be
low (O(n log n) version):¥n", T);
    printf("i¥tSA[i]¥tSuffix¥n");
    for (int i = 0; i < n; i++) printf("%2d¥t%2d¥t%s¥n", i, S
A[i], T + SA[i]);
    computeLCP(); // O(n)
    // LRS demo
    pii ans = LRS(); // find the LRS of the first input strin
g
    char lrsans[MAX_N];
    strncpy(lrsans, T + SA[ans.second], ans.first);
    printf("¥nThe LRS is '%s' with length = %d¥n¥n", lrsans,
ans.first);
    // stringMatching demo
    //printf("¥nNow, enter a string P below, we will try to f
ind P in T:¥n");
    strcpy(P, "A");
    m = (int) strlen(P);
    // if '¥n' is read, uncomment the next line
    //P[m-1] = 0; m--;
    pii pos = stringMatching();
    if (pos.first != -1 && pos.second != -1) {
        printf("%s is found SA[%d..%d] of %s¥n", P, pos.first
, pos.second, T);
        printf("They are:¥n");
        for (int i = pos.first; i <= pos.second; i++)
            printf("  %s¥n", T + SA[i]);
    } else printf("%s is not found in %s¥n", P, T);

    // LCS demo
    //printf("¥nRemember, T = '%s'¥nNow, enter another string
P:¥n", T);
    // T already has '$' at the back
    strcpy(P, "CATA");
    m = (int) strlen(P);
    // if '¥n' is read, uncomment the next line
    //P[m-1] = 0; m--;
    strcat(T, P); // append P
    strcat(T, "#"); // add '$' at the back
    n = (int) strlen(T); // update n

    // reconstruct SA of the combined strings
    constructSA(); // O(n log n)
    computeLCP(); // O(n)
    printf("¥nThe LCP information of 'T+P' = '%s':¥n", T);
    printf("i¥tSA[i]¥tLCP[i]¥tOwner¥tSuffix¥n");
    for (int i = 0; i < n; i++)
        printf("%2d¥t%2d¥t%2d¥t%2d¥t%s¥n", i, SA[i], LCP[i],
owner(SA[i]), T + SA[i]);
    ans = LCS(); // find the longest common substring between
T and P
    char lcsans[MAX_N];
    strncpy(lcsans, T + SA[ans.second], ans.first);
    printf("¥nThe LCS is '%s' with length = %d¥n", lcsans, an
s.first);
    return 0;
}
```

GEOMETRY:

Misc Geometric Formula:

Triangle	Circum Radius = a*b*c/(4*area) In Radius = area/s, where s = (a+b+c)/2 length of median to side c = sqrt(2*(a*a+b*b)-c*c)/2 length of bisector of angle C = sqrt(ab[(a+b)*(a+b)-c*c])/(a+b)
Ellipse	Area = PI*a*b Circumference = 4a *int(0,PI/2){sqrt(1-(k*sint)*(k*sint))}dt = 2*PI*sqrt((a*a+b*b)/2) approx where k = sqrt((a*a-b*b)/a) = PI*(3*(r1+r2)-sqrt[(r1+3*r2)*(3*r1+r2)])
Spherical cap	V = (1/3)*PI*h*h*(3*r-h) Surface Area = 2*PI*r*h
Spherical Sector	V = (2/3)*PI*r*r*h
Spherical Segment	V = (1/6)*PI*h*(3*a*a+3*b*b+h*h)
Torus	V = 2*PI*PI*R*r*r
Truncated Conic	V = (1/3)*PI*h*(a*a+a*b+b*b) Surface Area = PI*(a+b)*sqrt(h*h+(b-a)*(b-a)) = PI*(a+b)*l
Pyramidal frustum	(1/3)*h*(A1+A2+sqrt(A1*A2))

Misc Trigonometric Functions and Formulas:

tan A/2 = ±sqrt((1-cos A)/(1+cos A))
= sin A / (1+cos A)
= (1-cos A) / sin A
= cosec A – cot A

sin 3A = 3*sin A – 4*sincube A
cos 3A = 4*coscube A – 3*cos A
tan 3A = (3*tan A-tancube A)/(1-3*tansq A)
sin 4A = 4*sin A*cos A – 8*sincube A*cos A
cos 4A = $8*\cos^4 A - 8*\text{cossq } A + 1$
$[r*(\text{cost}+i*\text{sint})]^p = r^p*(\cos pt+i*\sin pt)$
acos**x** + **b**sin**x** **= c,** $x = 2n\pi + \alpha \pm \beta$, where
$\cos\alpha$ = a / (sqrt(a^2+b^2)), $\cos\beta$ = c / (sqrt(a^2+b^2));

2sinAcosB = sin(A+B) + sin(A-B)
2cosAsinB = sin(A+B) - sin(A-B)
2cosAcosB = cos(A-B) + cos(A+B)
2sinAsinB = cos(A-B) - cos(A+B)
sinC + sinD = 2sin[(C+D)/2]cos[(C-D)/2]
sinC - sinD = 2cos[(C+D)/2]sin[(C-D)/2]
cosD + cosC = 2cos[(C+D)/2]cos[(C-D)/2]
cosD - cosC = 2sin[(C+D)/2]sin[(C-D)/2]

Misc Integration Formula:

a^x => a^x/ln(a)
1/sqrt(x*x+a*a) => ln(x+sqrt(x*x+a*a))
1/sqrt(x*x-a*a) => ln(x+sqrt(x*x-a*a))
1/(x*sqrt(x*x+a*a) => -(1/a)*ln([a+sqrt(x*x+a*a)]/x)
1/(x*sqrt(a*a-x*x) => -(1/a)*ln([a+sqrt(a*a-x*x)]/x)

Misc Differentiation Formula:

asin x => 1/sqrt(1-x*x) acos x => -1/sqrt(1-x*x)
atan x => 1/(1+x*x) acot x => -1/(1+x*x)
asec x => 1/[x*sqrt(x*x-1)] acosec x => -1/[x*sqrt(x*x-1)]
a^x => a^x*ln(x) cot x => -cosecsq x
sec x => sec x * tan x cosec x => -cosec x * cot x

Mirror point(mx,my) of a point(x,y) w.r.to a line(ax+by+c=0):

```
void mirrorPoint(double a, double b, double c, double x, doub
le y, double & mx, double & my) {
    mx = x * (a * a - b * b) - 2.0 * a * b * y - 2.0 * a * c;
    mx /= (a * a + b * b);
    my = y * (a * a - b * b) - 2.0 * a * b * x - 2.0 * b * c;
    my /= (a * a + b * b);
}
```

Determining if a point lies on the interior of a 3D convex polygon:

```
// To determine whether a point is on the interior of a conve
x polygon in 3D, one
// might be tempted to first determine whether the point is o
n the plane, then
// determine its interior status. Both of these can be accomp
lished at once by
// computing the sum of the angles between the test point (q
below) and every pair of
```

```
// edge points p[i]->p[i+1]. This sum will only be twopi if both the point is on the
// plane of the polygon AND on the interior. The angle sum will tend to 0 the further
// away from the polygon point q becomes. The following code snippet returns the angle
// sum between the test point q and all the vertex pairs. The angle sum is in radians.
#define EPSILON 0.0000001
#define MODULUS(p) (sqrt(p.x * p.x + p.y * p.y + p.z * p.z))
const double TWOPI = 6.2831853071795864769252867,
    RTOD = 57.2957795;
double CalcAngleSum(point3D q, point3D * p, int n) {
    double m1, m2, anglesum = 0, costheta;
    point3D p1, p2;
    for (int i = 0; i < n; i++) {
        p1.x = p[i].x - q.x;
        p1.y = p[i].y - q.y;
        p1.z = p[i].z - q.z;
        p2.x = p[(i + 1) % n].x - q.x;
        p2.y = p[(i + 1) % n].y - q.y;
        p2.z = p[(i + 1) % n].z - q.z;
        m1 = MODULUS(p1), m2 = MODULUS(p2);
        if (m1 * m2 <= EPSILON) return (TWOPI); // We are on a node, consider this inside
        else costheta = (p1.x * p2.x + p1.y * p2.y + p1.z * p2.z) / (m1 * m2);
        anglesum += acos(costheta);
    }
    return (anglesum);
}
```

MISC GEOMETRY:

```
const double eps = 1e-11, pi = 2 * acos(0.0);
struct point { // Creates normal 2D point
    double x, y;
    point() {}
    point(double xx, double yy) {
        x = xx, y = yy;
    }
};
struct point3D { // Creates normal 3D point
    double x, y, z;
};
struct line { // Creates a line with equation ax + by + c = 0
    double a, b, c;
    line() {}
    line(point p1, point p2) {
        a = p1.y - p2.y;
        b = p2.x - p1.x;
        c = p1.x * p2.y - p2.x * p1.y;
    }
};
struct circle { // Creates a circle with point 'center' as center and r as radius
    point center;
    double r;
    circle() {}
    circle(point P, double rr) {
        center = P;
        r = rr;
    }
};
struct segment { // Creates a segment with two end points -> A, B
    point A, B;
    segment() {}
    segment(point P1, point P2) {
        A = P1, B = P2;
    }
};
inline bool eq(double a, double b) {
    return fabs(a - b) < eps;
} //two numbers are equal
```

Distance (Point, Point):

```
inline double Distance(point a, point b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
```

Distance ^ 2 (Point, Point):

```
inline double sq_Distance(point a, point b) {
    return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}
```

Distance (Point, Line):

```
inline double Distance(point P, line L) {
    return fabs(L.a * P.x + L.b * P.y + L.c) / sqrt(L.a * L.a + L.b * L.b);
}
```

Distance (Point, Segment):

```
inline double Distance(point P, segment S) {
    line L1 = line(S.A, S.B), L2;
    point P1;
    L2 = findPerpendicularLine(L1, P);
    if (intersection(L1, L2, P1))
        if (eq(Distance(S.A, P1) + Distance(S.B, P1), Distance(S.A, S.B)))
            return Distance(P, L1);
    return min(Distance(S.A, P), Distance(S.B, P));
}
```

IS Left Function:

```
inline double isleft(point p0, point p1, point p2) {
    return ((p1.x - p0.x) * (p2.y - p0.y) - (p2.x - p0.x) * (p1.y - p0.y));
}
```

Intersection (Line, Line):

```
inline bool intersection(line L1, line L2, point & p) {
    double det = L1.a * L2.b - L1.b * L2.a;
    if (eq(det, 0)) return false;
    p.x = (L1.b * L2.c - L2.b * L1.c) / det;
    p.y = (L1.c * L2.a - L2.c * L1.a) / det;
    return true;
}
```

Intersection (Segment, Segment):

```
inline bool intersection(segment L1, segment L2, point & p) {
    if (!intersection(line(L1.A, L1.B), line(L2.A, L2.B), p)) {
        return false; // can lie on another, just check their equations, and check overlap
    }
    return (eq(Distance(L1.A, p) + Distance(L1.B, p), Distance(L1.A, L1.B)) &&
        eq(Distance(L2.A, p) + Distance(L2.B, p), Distance(L2.A, L2.B)));
}
```

Perpendicular Line of a Given Line Through a Point:

```
inline line findPerpendicularLine(line L, point P) {
    line res; //line perpendicular to L, and intersects with P
    res.a = L.b, res.b = -L.a;
    res.c = -res.a * P.x - res.b * P.y;
    return res;
}
```

Area of a 2 D Polygon:

```
double areaPolygon(point P[], int n) {
    double area = 0;
    for (int i = 0, j = n - 1; i < n; j = i++) area += P[j].x * P[i].y - P[j].y * P[i].x;
    return fabs(area) / 2;
}
```

Point Inside Polygon:

```
bool insidePoly(point & p, point P[], int n) {
    bool inside = false;
    for (int i = 0, j = n - 1; i < n; j = i++)
        if (((P[i].x < p.x) ^ (P[j].x < p.x)) &&
```

```
                (P[i].y - P[j].y) * abs(p.x - P[j].x) < (p.y - P[j].y) * abs(P[i].x - P[j].x))
                inside = !inside;
        return inside;
    }
```

Intersection - Circle, Line:

```
inline bool intersection(circle C, line L, point & p1, point & p2) {
    if (Distance(C.center, L) > C.r + eps) return false;
    double a, b, c, d, x = C.center.x, y = C.center.y;
    d = C.r * C.r - x * x - y * y;
    if (eq(L.a, 0)) {
        p1.y = p2.y = -L.c / L.b;
        a = 1;
        b = 2 * x;
        c = p1.y * p1.y - 2 * p1.y * y - d;
        d = b * b - 4 * a * c;
        d = sqrt(fabs(d));
        p1.x = (b + d) / (2 * a);
        p2.x = (b - d) / (2 * a);
    } else {
        a = L.a * L.a + L.b * L.b;
        b = 2 * (L.a * L.a * y - L.b * L.c - L.a * L.b * x);
        c = L.c * L.c + 2 * L.a * L.c * x - L.a * L.a * d;
        d = b * b - 4 * a * c;
        d = sqrt(fabs(d));
        p1.y = (b + d) / (2 * a);
        p2.y = (b - d) / (2 * a);
        p1.x = (-L.b * p1.y - L.c) / L.a;
        p2.x = (-L.b * p2.y - L.c) / L.a;
    }
    return true;
}
```

Find Points that are r1 unit away from A, and r2 unit away from B:

```
inline bool findpointAr1Br2(point A, double r1, point B, double r2, point & p1, point & p2) {
    line L;
    circle C;
    L.a = 2 * (B.x - A.x);
    L.b = 2 * (B.y - A.y);
    L.c = A.x * A.x + A.y * A.y - B.x * B.x - B.y * B.y + r2 * r2 - r1 * r1;
    C.center = A;
    C.r = r1;
    return intersection(C, L, p1, p2);
}
```

Intersection Area between Two Circles:

```
inline double intersectionArea2C(circle C1, circle C2) {
    C2.center.x = Distance(C1.center, C2.center);
    C1.center.x = C1.center.y = C2.center.y = 0;
    if (C1.r < C2.center.x - C2.r + eps) return 0;
    if (-C1.r + eps > C2.center.x - C2.r) return pi * C1.r * C1.r;
    if (C1.r + eps > C2.center.x + C2.r) return pi * C2.r * C2.r;
    double c, CAD, CBD, res;
    c = C2.center.x;
    CAD = 2 * acos((C1.r * C1.r + c * c - C2.r * C2.r) / (2 * C1.r * c));
    CBD = 2 * acos((C2.r * C2.r + c * c - C1.r * C1.r) / (2 * C2.r * c));
    res = C1.r * C1.r * (CAD - sin(CAD)) + C2.r * C2.r * (CBD - sin(CBD));
    return .5 * res;
}
```

Circle Through Thee Points:

```
circle CircleThrough3points(point A, point B, point C) {
    double den;
    circle c;
    den = 2.0 * ((B.x - A.x) * (C.y - A.y) - (B.y - A.y) * (C.x - A.x));
    c.center.x = ((C.y - A.y) * (B.x * B.x + B.y * B.y - A.x * A.x - A.y * A.y)-
            (B.y - A.y) * (C.x * C.x + C.y * C.y - A.x * A.x - A.y * A.y));
    c.center.x /= den;
    c.center.y = ((B.x - A.x) * (C.x * C.x + C.y * C.y - A.x * A.x - A.y * A.y)-
            (C.x - A.x) * (B.x * B.x + B.y * B.y - A.x * A.x - A.y * A.y));
    c.center.y /= den;
    c.r = Distance(c.center, A);
    return c;
}
```

Rotating a Point anticlockwise by 'theta' radian w.r.t Origin:

```
inline point rotate2D(double theta, point P) {
    point Q;
    Q.x = P.x * cos(theta) - P.y * sin(theta);
    Q.y = P.x * sin(theta) + P.y * cos(theta);
    return Q;
}
```

Convex Hull(Graham Scan) O(nlogn):

```
// compare Function for qsort in convex hull
point Firstpoint;
int cmp(const void * a,
    const void * b) {
    double x, y;
    point aa, bb;
    aa = * (point * ) a;
    bb = * (point * ) b;
    x = isleft(Firstpoint, aa, bb);
    if (x > eps) return -1;
    else if (x < -eps) return 1;
    x = sq_Distance(Firstpoint, aa);
    y = sq_Distance(Firstpoint, bb);
    if (x + eps < y) return -1;
    return 1;
}
// 'P' contains all the points, 'C' contains the convex hull
// 'nP' = total points of 'P', 'nC' = total points of 'C'
void ConvexHull(point P[], point C[], int & nP, int & nC) {
    int i, j, pos = 0; // Remove duplicate points if necesary
    for (i = 1; i < nP; i++)
        if (P[i].y < P[pos].y || (eq(P[i].y, P[pos].y) && P[i].x > P[pos].x + eps))
            pos = i;
    swap(P[pos], P[0]);
    Firstpoint = P[0];
    qsort(P + 1, nP - 1, sizeof(point), cmp);
    C[0] = P[0];
    C[1] = P[1];
    i = 2, j = 1;
    while (i < nP) {
        if (isleft(C[j - 1], C[j], P[i]) > -eps) C[++j] = P[i++];
        else j--;
    }
    nC = j + 1;
}
```

Angle between Vectors:

```
inline double angleBetweenVectors(point O, point A, point B) { // vector OA to OB
    point t1, t2;
    t1.x = A.x - O.x;
    t1.y = A.y - O.y;
    t2.x = B.x - O.x;
    t2.y = B.y - O.y;
    double theta = (atan2(t2.y, t2.x) - atan2(t1.y, t1.x));
    if (theta < 0) theta += 2 * pi;
    return theta;
}
```

MISC:

FORMULA:

1. Cayley's Formula: There are n^{n-2} spanning trees of a complete graph with n labeled vertices. Example: UVa 10843 - Anne's game.
2. Derangement: A permutation of the elements of a set such that none of the elements appear in their original position. The number of derangements $der(n)$ can be computed as follow: $der(n) = (n-1) \times (der(n-1) + der(n-2))$ where $der(0) = 1$ and $der(1) = 0$. A basic problem involving derangement is UVa 12024 - Hats (see Section 5.6).
3. Erdős Gallai's Theorem gives a necessary and sufficient condition for a finite sequence of natural numbers to be the *degree sequence* of a simple graph. A sequence of non-negative integers $d_1 \geq d_2 \geq \ldots \geq d_n$ can be the degree sequence of a simple graph on n vertices iff $\sum_{i=1}^{n} d_i$ is even and $\sum_{i=1}^{k} d_i \leq k \times (k-1) + \sum_{i=k+1}^{n} min(d_i, k)$ holds for $1 \leq k \leq n$. Example: UVa 10720 - Graph Construction.
4. Euler's Formula for Planar Graph[6]: $V - E + F = 2$, where F is the number of faces[7] of the Planar Graph. Example: UVa 10178 - Count the Faces.
5. Moser's Circle: Determine the number of pieces into which a circle is divided if n points on its circumference are joined by chords with no three internally concurrent. Solution: $g(n) = {}^{n}C_4 + {}^{n}C_2 + 1$. Example: UVa 10213 - How Many Pieces of Land?
6. Pick's Theorem[8]: Let I be the number of integer points in the polygon, A be the area of the polygon, and b be the number of integer points on the boundary, then $A = i + \frac{b}{2} - 1$. Example: UVa 10088 - Trees on My Island.
7. The number of spanning tree of a complete bipartite graph $K_{n,m}$ is $m^{n-1} \times n^{m-1}$. Example: UVa 11719 - Gridlands Airport.

CATALAN NUMBER PROPERTIES:

1. $Cat(n)$ counts the number of distinct binary trees with n vertices, e.g. for $n = 3$:

2. $Cat(n)$ counts the number of expressions containing n pairs of parentheses which are correctly matched, e.g. for $n = 3$, we have: ()()(), ()(()), (())(), ((())), and (()()).
3. $Cat(n)$ counts the number of different ways $n + 1$ factors can be completely parenthesized, e.g. for $n = 3$ and $3 + 1 = 4$ factors: {a, b, c, d}, we have: (ab)(cd), a(b(cd)), ((ab)c)d, (a(bc))(d), and a((bc)d).
4. $Cat(n)$ counts the number of ways a convex polygon (see Section 7.3) of $n + 2$ sides can be triangulated. See Figure 5.1, left.
5. $Cat(n)$ counts the number of monotonic paths along the edges of an $n \times n$ grid, which do not pass above the diagonal. A monotonic path is one which starts in the lower left corner, finishes in the upper right corner, and consists entirely of edges pointing rightwards or upwards. See Figure 5.1, right and also see Section 4.7.1.

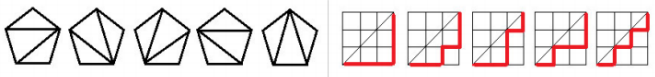


Figure 5.1: Left: Triangulation of a Convex Polygon, Right: Monotonic Paths

Nth Permutation:

```
const int NX = 30;
Long fact[NX], nth_value;
int freq[NX];
char inp[NX];
void ini() {
    fact[0] = 1;
    Long i;
    for (i = 1; i <= 25; i++) fact[i] = (fact[i - 1] * i);
}
Long occurence(int len) {
    Long now = fact[len];
    int i;
    for (i = 0; i < 26; i++) now /= fact[freq[i]];
    return now;
}
void solve(int sz) {
    while (sz) {
        Long upto = 0;
        bool found = 0;
        int i;
        for (i = 0; i < 26 && !found; i++) {
            if (freq[i] == 0) continue;
            freq[i]--;
            Long now = occurence(sz - 1);
            if (now + upto >= nth_value) {
                nth_value -= upto;
                found = 1;
                printf("%c", i + 'a');
                sz--;
            } else {
                upto += now;
                freq[i]++;
            }
            // printf("i :: %c sz :: %d now :: %lld upto :: %lld nth :: %lld ¥n", i+'a', sz, now, upto, nth_value);

        }
        if (!found) break;
    }
    puts("");
}
int main() {
    int cs, t;
    getI(t);
    ini();
    for (cs = 1; cs <= t; cs++) {
        scanf("%s %lld", inp, & nth_value);
        ms(freq, 0);
        int sz = strlen(inp);
        // rep( i , sz ) printf(" integer :: %d ¥n", inp[i]-'a');
        rep(i, sz) freq[inp[i] - 'a']++;
        Long need = occurence(sz);
        printf("Case %d: ", cs);
        if (need < nth_value) puts("Impossible");
        else solve(sz);
    }
}
```

BACKTRACKING (N QUEEN PROBLEM):

```
int board[8][8];
int soln[100][8];
int tot = 0;
bool isSafe(int row, int col) {
    int i, j;
    for (i = 0; i < 8; i++) {
        if (board[row][i] && i != col)
            return false;
    }
    for (int i = 0; i < 8; i++) {
        if (i == row) continue;
        for (int j = 0; j < 8; j++) {
            if (j == col) continue;
            if (abs(row - i) == abs(col - j) && board[i][j])
                return false;
        }
    }
    return true;
}
void solveNQUtil(int col) {
    if (col == 8) {
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {
                if (board[i][j])
                    soln[tot][i] = j;
            }
        }
        tot++;
        return;
    }
    for (int i = 0; i < 8; i++) {
        if (isSafe(i, col)) {
            board[i][col] = 1;
            solveNQUtil(col + 1);
            board[i][col] = 0;
        }
    }
}
```

STRTOK:

```
int main ()
{
    char str[] ="- This, a sample string.";
    char * pch;
```

```
    pch = strtok (str," ,.-
");          // by which charcters string is tokenized that s
hould be given into "(here)"
    while (pch != NULL)
    {
        printf ("%s¥n",pch);
        pch = strtok (NULL, " ,.-
");    // by which charcters string is tokenized that should
be given into "(here)"
    }
    return 0;
}
```

BITWISE OPERATOR:

```
// Odd - Even checking ==>>
if(x & 1) --> Odd
else --> Even
// 2^n dara gun or vag ==>>
gun --> x<<n
vag --> x>>n
// 2^n or 2 er power kina ==>>
if(x & (x-1)) --> 2 er power na
else --> 2 er power
// 2^n dara divisible naki ==>>
let, d=2^n
int x,d=8; // 8=2^3
if(x & (d-1)) --> x,d dara divisible
else --> x,d dara divisible na
//SWAP ==>>
int x,y;
x = x ^ y;
y = x ^ y;
x = x ^ y;
```

PERMUTATION & COMBINATION UPTO 30:

```
ll dp_permu[31][31], dp_combi[31][31];
void permu() {
    for (int i = 0; i <= 30; i++) {
        dp_permu[i][0] = 1;
        dp_permu[i][1] = i;

        for (int j = 2; j <= i; j++)
            dp_permu[i][j] = dp_permu[i][j - 1] * (ll)(i - j
+ 1);
    }
}
void combi() {
    for (int i = 0; i < 31; i++) {
        dp_combi[i][0] = dp_combi[i][i] = 1;
        for (int j = 1; j < i; j++) {
            dp_combi[i][j] = dp_combi[i - 1][j - 1] + dp_comb
i[i - 1][j];
        }
    }
}
```

JOSEPHUS PROBLEM:

```
int joseph(int n, int k) {
    if (n == 1) return 0;
    return ((joseph(n - 1, k) + k) % n);
}
```

SOME NOTES:

- Gray Code □ □□ □□□ □ change হয় তার পর 1000…. থাকে
- Segment tree **এর সময়** 2 **পাশরে** tree **এর** marge check **করতে হয়।**
- BIT **সংক্রান্ত কছিু হলে** TRIE **কনিা দখেতে হবে**
- Central Bionomial Coefficient =
  (2n, n) = (n!) / ((n-k)! * (k!))
  N -> 3 = 20, 4 = 70, 5 = 252, 6 = 924, 7 = 3432

mamun4122:

```
/* Given a number N, let d be a divisor of N. Then the number
of pairs {a,N}, where 1≤a≤N and gcd(a,N)=d, is φ(N/d)
** Approximate number of primes under n= (n/ln(n))
** Approximate upper limit of number of divisor =2∛N
** Diphonite eqn gulai negative number niye hisab korte hbe
** Once we find a pair (x,y) using ext_gcd, we can generate i
nfinite pairs of Bezout coefficients using the formula: (x+(k
*b)/gcd(a,b),y-(k*a)/gcd(a,b))
** Goldbach' s Conjecture: For any integer n (n ≥ 4) there ex
ist two prime numbers p1 and p2 such that p1 + p2 = n.
** For a given positive integer n (0 < n < 231) we need to fi
nd the number of such m that 1 ≤ m ≤ n, GCD(m, n) ≠ 1 and GC
D(m, n) ≠ m
n - φ(n) - (a1 + 1) * (a2 + 1) * … * (ak + 1) + 1 */
```

dipta007:

```
/*Area of a triangle :
Let K be the triangle's area and let a, b and c, be the lengt
hs of its sides. By Heron' s Formula , the area of the trian
gle is K = sqrt( s * (s-a) * (s-b) * (s -c) ).
where S is the semiperimeter s = (a+b+c)/2.
** length of median to side c = sqrt(2*(a*a+b*b)-c*c)/2
** length of bisector of angle C =  sqrt(ab[(a+b)*(a+b)-
c*c])/(a+b)
** Radius of a In-
cicle: The radius of the incircle is r = (2*k)/P = sqrt((s-
a)*(s-b)*(s-
c)/s). Thus, the area K of a triangle may be found by multipl
ying the inradius by the semiperimeter: K = rs.
** Angle : For a regular convex n-
gon, each interior angle has a measure of: (n-
2)*180/n degrees.
** Apothem: The apothem of a regular polygon is a line segmen
t from the center to the midpoint of one of its sides.Equival
ently, it is the line drawn from the center of the polygon th
at is perpendicular to one of its sides.
** Circumradius: The circumradius from the center of a regula
r polygon to one of the vertices is related to the side lengt
h s or to the apothem a by r = s/(2sin(PI/n)) = a/(cos(PI/n))

** Area : The area A of a convex regular n-
sided polygon having Side s, circumradius r, apothem a, and p
erimeter p is given by */
```

$$A = \frac{1}{2}nsa = \frac{1}{2}pa = \frac{1}{4}ns^2\cot\frac{\pi}{n} = na^2\tan\frac{\pi}{n} = \frac{1}{2}nr^2\sin\frac{2\pi}{n}$$

howcum:

```
/* Hockeystick pattern: (2c2 * 3c2 * ……. * (n+1)c2) = (n+2)
c(2+1)
** Equation for the angle of the hour hand: angle_hour= (½) *
(60H + M)
** Equation for the angle of the minute hand: angle_minute =
6M
** Equation for the angle of the both hand : angle = abs(angl
e_hour - angle_minute) */
```

PALINDROMIC INDEX:

```
/* The position of a palindrome within the sequence can be de
termined almost without calculation: If the palindrome has an
even number of digits, prepend a 1 to the front half of the p
alindrome's digits. If the number of digits is odd, prepend t
he value of front digit + 1 to the digits from position 2 ...
central digit. Examples: 98766789=a(19876), 515=a(61), 820602
8=a(9206), 9230329=a(10230). */
```

TERNARY SEARCH:

```
double ts() {
    double min = 0;
    double max = 1;
    int c = 100; //for higher precision have to increase
    double k, l, f, g;
    while (c--) {
        f = min + (max - min) / (double) 3.0;
        g = min + (double) 2.0 * ((max - min) / (double) 3.0)
;
        k = fun(f);
        l = fun(g);
        if (k < l) {
            max = g;
```

```
13.         } else {
14.             min = f;
15.         }
16.     }
17.     return (min + max) / 2.0;
18. }
```

TEMPLATE:

```
#pragma comment(linker, "/stack:640000000")

#include <algorithm>
#include <bitset>
#include <cassert>
#include <cctype>
#include <climits>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <fstream>
#include <iostream>
#include <iomanip>
#include <iterator>
#include <list>
#include <map>
#include <numeric>
#include <queue>
#include <set>
#include <sstream>
#include <stack>
#include <string>
#include <utility>
#include <vector>
using namespace std;

const double EPS = 1e-9;
const int INF = 0x7f7f7f7f;
const double PI=acos(-1.0);

#define    READ(f)            freopen(f, "r", stdin)
#define    WRITE(f)           freopen(f, "w", stdout)
#define    MP(x, y)           make_pair(x, y)
#define    PB(x)              push_back(x)
#define    rep(i,n)           for(int i = 1 ; i<=(n) ; i++)
#define    repI(i,n)          for(int i = 0 ; i<(n) ; i++)
#define    FOR(i,L,R)         for (int i = L; i <= R; i++)
#define    ROF(i,L,R)         for (int i = L; i >= R; i--)
#define    FOREACH(i,t)       for (typeof(t.begin()) i=t.begin(); i!=t.end(); i++)
#define    ALL(p)             p.begin(),p.end()
#define    ALLR(p)            p.rbegin(),p.rend()
#define    SET(p)             memset(p, -1, sizeof(p))
#define    CLR(p)             memset(p, 0, sizeof(p))
#define    MEM(p, v)          memset(p, v, sizeof(p))
#define    getI(a)            scanf("%d", &a)
#define    getII(a,b)         scanf("%d%d", &a, &b)
#define    getIII(a,b,c)      scanf("%d%d%d", &a, &b, &c)
#define    getL(a)            scanf("%lld",&a)
#define    getLL(a,b)         scanf("%lld%lld",&a,&b)
#define    getLLL(a,b,c)      scanf("%lld%lld%lld",&a,&b,&c)
#define    getC(n)            scanf("%c",&n)
#define    getF(n)            scanf("%lf",&n)
#define    getS(n)            scanf("%s",n)
#define    bitCheck(a,k)      ((bool)(a&(1<<(k))))
#define    bitOff(a,k)        (a&(~(1<<(k))))
#define    bitOn(a,k)         (a|(1<<(k)))
#define    iseq(a,b)          (fabs(a-b)<EPS)
#define    vi    vector < int >
#define    vii   vector < vector < int > >
#define    pii   pair< int, int >
#define    ff    first
#define    ss    second
#define    ll    long long
#define    ull   unsigned long long

template< class T > inline T _abs(T n) { return ((n) < 0 ? -(n) : (n)); }
template< class T > inline T _max(T a, T b) { return (!((a)<(b))?(a):(b)); }
template< class T > inline T _min(T a, T b) { return (((a)<(b))?(a):(b)); }
template< class T > inline T _swap(T &a, T &b) { a=a^b;b=a^b;a=a^b;}
template< class T > inline T gcd(T a, T b) { return (b) == 0 ? (a) : gcd((b), ((a) % (b))); }
template< class T > inline T lcm(T a, T b) { return ((a) / gcd((a), (b)) * (b)); }
template <typename T> string NumberToString ( T Number ) { ostringstream ss; ss << Number; return ss.str(); }

#ifdef CSE13
     #define debug(args...) {cerr<<"*: "; dbg,args; cerr<<endl;}
#else
    #define debug(args...)  // Just strip off all debug tokens
#endif

struct debugger{
    template<typename T> debugger& operator , (const T& v){
        cerr<<v<<" ";
        return *this;
    }
}dbg;
///****************** template ends here ***************
int t,n,m;

int main() {
    ///check for 0 or -1 if input not specified
    #ifdef CSE13
//        READ("in.txt");
//        WRITE("out.txt");
    #endif // mamun
}
//    clock_t begin, end;
//    double time_spent;
//    begin = clock();
//
//    end = clock();
//    time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
//    cerr<<"Time spent = "<<time_spent<<endl;
```