Check if a given point lies inside or outside a polygon

#include <bits/stdc++.h>

using namespace std;

struct Point {

int x, y;

};

const int INF = 10000;

// check if point q lies on line segment 'pr'

bool onSegment(Point p, Point q, Point r) {

if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) && q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))

return true;

return false;

}

int orientation(Point p, Point q, Point r) {

int val = (q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y);

if (val == 0)

return 0;

return (val > 0) ? 1 : 2;

}

// Check if line segment 'p1q1' and 'p2q2' intersect

bool doIntersect(Point p1, Point q1, Point p2, Point q2) {

int o1 = orientation(p1, q1, p2);

int o2 = orientation(p1, q1, q2);

int o3 = orientation(p2, q2, p1);

int o4 = orientation(p2, q2, q1);

if (o1 != o2 && o3 != o4)

return true;

if (o1 == 0 && onSegment(p1, p2, q1))

return true;

if (o2 == 0 && onSegment(p1, q2, q1))

return true;

if (o3 == 0 && onSegment(p2, p1, q2))

return true;

if (o4 == 0 && onSegment(p2, q1, q2))

return true;

return false;

}

// Check if a given point lies inside ot outside a polygon

bool isInside(Point polygon[], int n, Point p) {

// There must be at least 3 vertices in polygon[]

if (n < 3)

return false;

// Create a point for line segment from p to infinite

Point extreme = { INF, p.y };

// Count intersections of the above line with sides of polygon

int count = 0, i = 0;

do {

int next = (i + 1) % n;

// Check if the line segment from 'p' to 'extreme' intersects

// with the line segment from 'polygon[i]' to 'polygon[next]'

if (doIntersect(polygon[i], polygon[next], p, extreme)) {

// If the point 'p' is colinear with line segment 'i-next',

// then check if it lies on segment. If it lies, return true,

// otherwise false

if (orientation(polygon[i], p, polygon[next]) == 0)

return onSegment(polygon[i], p, polygon[next]);

count++;

}

i = next;

} while (i != 0);

return count & 1;

}

Check if two given line segments intersect

include <bits/stdc++.h>

using namespace std;

struct Point {

int x, y;

};

// check if point q lies on line segment 'pr'

bool onSegment(Point &p, Point &q, Point &r) {

if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) && q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))

return true;

return false;

}

// To find orientation of ordered triplet (p, q, r).

// The function returns following values

// 0 --> p, q and r are colinear

// 1 --> Clockwise

// 2 --> Counterclockwise

int orientation(Point p, Point q, Point r) {

int val = (q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y);

if (val == 0)

return 0; // colinear

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

}

bool doIntersect(Point p1, Point q1, Point p2, Point q2) {

int o1 = orientation(p1, q1, p2);

int o2 = orientation(p1, q1, q2);

int o3 = orientation(p2, q2, p1);

int o4 = orientation(p2, q2, q1);

// General case

if (o1 != o2 && o3 != o4)

return true;

// p1, q1 and p2 are colinear and p2 lies on segment p1q1

if (o1 == 0 && onSegment(p1, p2, q1))

return true;

// p1, q1 and p2 are colinear and q2 lies on segment p1q1

if (o2 == 0 && onSegment(p1, q2, q1))

return true;

// p2, q2 and p1 are colinear and p1 lies on segment p2q2

if (o3 == 0 && onSegment(p2, p1, q2))

return true;

// p2, q2 and q1 are colinear and q1 lies on segment p2q2

if (o4 == 0 && onSegment(p2, q1, q2))

return true;

return false; // Doesn't fall in any of the above cases

}

// Check if a given point lies inside ot outside a polygon

bool isInside(Point polygon[], int n, Point p) {

// There must be at least 3 vertices in polygon[]

if (n < 3)

return false;

// Create a point for line segment from p to infinite

Point extreme = { INF, p.y };

// Count intersections of the above line with sides of polygon

int count = 0, i = 0;

do {

int next = (i + 1) % n;

// Check if the line segment from 'p' to 'extreme' intersects

// with the line segment from 'polygon[i]' to 'polygon[next]'

if (doIntersect(polygon[i], polygon[next], p, extreme)) {

// If the point 'p' is colinear with line segment 'i-next',

// then check if it lies on segment. If it lies, return true,

// otherwise false

if (orientation(polygon[i], p, polygon[next]) == 0)

return onSegment(polygon[i], p, polygon[next]);

count++;

}

i = next;

} while (i != 0);

return count & 1;

}

Find Simple Closed Path for a given set of all points

#include <bits/stdc++.h>

using namespace std;

struct Point {

int x, y;

} p0;

int dist(Point p1, Point p2) {

return (p1.x - p2.x) \* (p1.x - p2.x) + (p1.y - p2.y) \* (p1.y - p2.y);

}

int orientation(Point p, Point q, Point r) {

int val = (q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y);

if (val == 0)

return 0;

return (val > 0) ? 1 : 2;

}

int compare(const void \*vp1, const void \*vp2) {

Point \*p1 = (Point \*) vp1;

Point \*p2 = (Point \*) vp2;

int o = orientation(p0, \*p1, \*p2);

if (o == 0)

return (dist(p0, \*p2) >= dist(p0, \*p1)) ? -1 : 1;

return (o == 2) ? -1 : 1;

}

void printClosedPath(Point points[], int n) {

// Find the bottommost point

int ymin = points[0].y, min = 0;

for (int i = 1; i < n; i++) {

int y = points[i].y;

// Pick the bottom-most. In case of tie, chose the

// left most point

if ((y < ymin) || (ymin == y && points[i].x < points[min].x))

ymin = points[i].y, min = i;

}

// Place the bottom-most point at first position

swap(points[0], points[min]);

// Sort n-1 points with respect to the first point.

// A point p1 comes before p2 in sorted ouput if p2

// has larger polar angle (in counterclockwise

// direction) than p1

p0 = points[0];

qsort(&points[1], n - 1, sizeof(Point), compare);

for (int i = 0; i < n; i++)

cout << "(" << points[i].x << ", " << points[i].y << "), ";

Minimum angle to see all given points

vector<double> v;

int main() {

// freopen("a", "r", stdin);

scanf("%d", &n);

for (int i = 0; i < n; ++i) {

int x, y;

scanf("%d%d", &x, &y);

v.push\_back(atan2(y \* 1.0, x \* 1.0));

}

sort(v.begin(), v.end());

double res = v[n - 1] - v[0];

for (int i = 1; i < n; ++i)

res = min(res, 2 \* PI - (v[i] - v[i - 1]));

printf("%.9lf\n", res \* 180 / PI);

return 0;

}

// convert 2 points to vector a- >b

vec toVec(point a, point b) {

return vec(b.x - a.x, b.y - a.y);

}

/\*

\* XOR for all integers in range from A to B inclusive

\*/

typedef long long ll;

inline ll calc(ll n) {

ll a[] = { n, 1, n + 1, 0 };

return a[n % 4];

}

int main() {

ll a, b;

while (scanf("%lld%lld", &a, &b) > 0)

printf("%lld\n", calc(b) ^ (calc(a - 1)));

return 0;

}

// Find the lexicographically 'K'th permutation of the sequence [1..N] .. O(N^2)

typedef long long ll;

const int N = 20;

int n, k;

ll fact[N];

bool vis[N];

vector<int> sol;

int main() {

scanf("%d%d", &n, &k);

fact[0] = 1;

for (int i = 1; i <= n; ++i)

fact[i] = fact[i - 1] \* i;

ll cur = 0;

for (int i = 1; i <= n; ++i)

for (int j = 1; j <= n; ++j) {

if (vis[j])

continue;

ll add = fact[n - i];

if (cur + add >= k) {

vis[j] = true;

sol.push\_back(j);

break;

}

cur += add;

}

for (int i = 0; i < n; ++i)

printf("%s%d", i ? " " : "", sol[i]);

puts("");

return 0;

}

GCD of two numbers when one of them can be very large

long long gcd(long long a, char \*b) {

int n = strlen(b);

long long mod = 0;

for (int i = 0; i < n; ++i)

mod = (mod \* 10 + (b[i] - '0')) % a;

return \_\_gcd(a, mod);

}

Matrix Op

typedef vector<vector<int> > mat;

// Matrix Clockwise Rotate

mat rotate(mat &m) {

mat res(m[0].size(), vector<int>(m.size()));

for (int i = 0; i < (int) m.size(); ++i)

for (int j = 0; j < (int) m[i].size(); ++j)

res[j][m.size() - 1 - i] = m[i][j];

return res;

}

// Matrix Reflection

mat reflect(mat &m) {

mat res(m.size(), vector<int>(m[0].size()));

for (int i = 0; i < (int) m.size(); ++i)

for (int j = 0; j < (int) m[i].size(); ++j)

res[i][m[i].size() - 1 - j] = m[i][j];

return res;

}

// Matrix Multiplication

mat mul(mat &a, mat &b) {

mat res(a.size(), vector<int>(b[0].size()));

for (int i = 0; i < (int) res.size(); ++i)

for (int j = 0; j < (int) res[i].size(); ++j)

for (int k = 0; k < (int) b.size(); ++k)

res[i][j] += a[i][k] \* b[k][j];

return res;

}

// Matrix Power

mat Pow(mat &base, int p) {

mat res(base.size(), vector<int>(base[0].size()));

for (int i = 0; i < (int) base.size(); ++i)

for (int j = 0; j < (int) base[i].size(); ++j)

if (i == j)

res[i][j] = 1;

while (p != 0) {

if (p & 1)

res = mul(res, base);

base = mul(base, base);

p >>= 1;

}

return res;

}

// Matrix Power - Using Recursion

mat Pow(mat &base, int p) {

if (p == 1)

return base;

mat res = Pow(res, p >> 1);

res = mul(res, res);

if (p & 1)

res = mul(res, base);

return res;

}

FUN INT FROM DOUBLE

// Compare doubles - return 0 for a==b, 1 for a>b, -1 for a < b

bool comp\_double(double &a, double &b) {

if (fabs(a - b) <= 1e-10)

return 0;

return a < b ? -1 : 1;

}

// ceil function

int ceil(int a, int b) {

return (a + b - 1) / b;

}

// round function

int round(int a, int b) {

if (a > 0)

return (a + b / 2) / b;

return (a - b / 2) / b;

}

// round to multiple of a specified amount

int round(int a, int m) {

return round(a / m) \* m;

}

// find number of digits in number

int numberOfDigits(int n, int base) {

double a = log10(n), b = log10(base);

return 1 + floor(a / b);

}

/\*

Only those numbers, which are perfect Squares have an odd number of factors.

\*/

/\*

\* Check if given number is Fibonacci number

\* A number n is Fibonacci if (5\*n^2 + 4) or (5\*n^2 - 4) is a perfect square

\*/

// Wilson's theorem: (p-1)!%p = p-1 IFF p is prime

/\*

\* a == b % m, and b == c % m, then a == c % m

\* a == b % m, and c == d % m, then a (+/-) c == (b (+/-) d) % m

\* a == b % m, and c == d % m, then ac == bd % m

\* a == b % m, then a + c == (b + c) % m

\* a == b % m, then ca == cb % m

\*/

Extra ..

const int N = 1e5 + 1;

bitset<N> bs;

int n, phi[N], mo[N];

/\*

\* Phi Code: Range Generator

\* Phi Function: Count integers i < n such that gcd(i, n) = 1

\*/

void PhiGenerator() {

memset(phi, 1, sizeof phi);

for (int i = 2; i < N; ++i)

if (!bs[i]) {

phi[i] = i - 1;

for (ll j = i \* 2; j < N; j += i) {

bs[j] = 1;

ll n = j, pow = 1;

while (n % i == 0) {

n /= i;

pow \*= i;

}

phi[j] \*= (pow / i) \* (i - 1);

}

}

}

// phi(N!) = (N is prime ? N - 1 : N ) \* phi((N - 1)!)

ll phi\_fact(int n) {

ll res = 1;

for (int i = 1; i <= n; ++i)

res \*= (bs[i] == 0) ? i - 1 : i;

return res;

}

// Mobius function µ

// µ(n) = 1 if n is a square-free positive integer with an even number of prime factors.

// µ(n) = -1 if n is a square-free positive integer with an odd number of prime factors.

// µ(n) = 0 if n has a squared prime factor.

void Mobius() {

fill(mo, mo + N, 1);

for (int i = 2; i < N; ++i)

if (!bs[i]) {

for (int j = i; j < N; j += i) {

bs[j] = 1;

mo[j] \*= -1;

}

ll p = (ll) i \* i;

for (ll j = p; j < N; j += p)

mo[j] = 0;

}

}

// Given square free number, find its index

int findIdx(int n) {

int idx = n;

for (int i = 2; i \* i <= n; ++i)

idx -= mo[i] \* (n / (i \* i));

return idx;

}

// Count the triples (a,b,c) such a, b, c <= n, and gcd(a, b, c) = 1

ll calc(int n) {

ll res = 1LL \* n \* n \* n;

for (int i = 2; i <= n; ++i)

res -= mo[i] \* (n / i) \* (n / i) \* (n / i);

return res;

}

int Pow(int x, int y, int mod) {

int res = 1;

while (y) {

if (y & 1)

res = (1LL \* res \* x) % mod;

x = (1LL \* x \* x) % mod;

y >>= 1;

}

return res;

}

// calculate a1^a2^a3^... ^aN mod m (m is not prime)

int mmpow(int idx, int A[], int m, int n) {

if (idx == n - 1)

return A[idx] % m;

int p = mmpow(idx + 1, A, phi[m], n) + phi[m];

return Pow(A[idx], p, m);

}

// compute all Mod inverse for range 1 - (p-1)

vector<int> ModInvRange(int p) {

vector<int> inv(p, 1);

for (int i = 2; i < p; ++i)

inv[i] = (p - (p / i) \* inv[p % i] % p) % p;

return inv;

}

// calculate (a^m) % p when m is very large

ll modive(ll a, ll m, ll p) {

return Pow(a, (m % (p - 1) \* (p - 2) % (p - 1)) % (p - 1), p);

}