#algorithms #study

정수론

소수 판별

- 소수 : 약수로 자기 자신과 1만을 가지는 수

```
bool isPrime(int n) {
    if (n <= 2)
        return false;
    if (n % 2 == 0)
        return false;
    int sqrtn = int(sqrt(n));
    for (int div = 3; div <= sqrtn; div += 2)
        if (n % div == 0)
            return false;
    return true;
}
```

소인수분해

- 소인수 분해 : 합성수를 소수들의 곱으로 나타낸 것

```
vector<int> factorSimple(int n) {
    vector<int> ret;
    int sqrtn = int(sqrt(n));
    for (int div = 2; div <= sqrtn; ++div) {
        while (n % div == 0) {
            n /= div;
            ret.push_back(div);
        }
    }
    if (n > 1)
        ret.push_back(n);
    return ret;
}
```

에라토스테네스의 체

- 에라토스테네스의 체 : n 이하의 소수를 찾아내는 방법

```
int n;
bool isPrime[MAX_N + 1];
```

```
void eratosthenes() {
    memset(isPrime, 1, sizeof(isPrime));
    isPrime[0] = isPrime[1] = false;
    int sqrtn = int(sqrt(n));
    for (int i = 2; i <= sqrtn; ++i) {
        if (isPrime[i])
            for (int j = i * i; j <= n; j += i)
                isPrime[j] = false;
    }
}
```

유클리드 알고리즘

- 유클리드 알고리즘 : 두 수의 최대공약수를 구하는 알고리즘

```
int gcd(int p, int q) {
    if (q == 0)
        return p;
    return gcd(q, p % q);
}
```

나머지 연산

- 나머지 연산이란 M 값에 도달하면 0으로 돌아가는 정수들을 가지고 진행하는 연산이다.
- 나머지 연산은 덧셈, 뺄셈, 곱셈에 관해서 아래와 같은 성질을 가진다
  - (a + b) % M = ((a % M) + (b % M)) % M
  - (a - b) % M = ((a % M) - (b % M)) % M
  - (a * b) % M = ((a % M) * (b % M)) % M

기하계산

정의

- 점, 선, 다각형과 원 등 각종 기하학적 도형을 다루는 알고리즘

벡터 구현

```
const double PI = 2.0 * acos(0.0); // 2차원 벡터 표현

struct vector2 {
    double x, y;

    explicit vector2(double x_ = 0, double y_ = 0) : x(x_), y(y_) {}

    bool operator==(const vector2& rhs) const {
```

```
        return x == rhs.x && y == rhs.y;
    }
    bool operator<(const vector2& rhs) const {
        return x != rhs.x ? x < rhs.x : y < rhs.y;
    }
    vector2 operator+(const vector2& rhs) const {
        return vector2(x + rhs.x, y + rhs.y);
    }
    vector2 operator-(const vector2& rhs) const {
        return vector2(x - rhs.x, y - rhs.y);
    }
    vector2 operator*(const vector2& rhs) const {
        return vector2(x * rhs, y * rhs);
    }
    // 벡터 길이 반환
    double norm() const { return hypot(x, y); }
    // 방향이 같은 단위 벡터 반환
    vector2 normalize() const {
        return vector2(x / norm(), y / norm());
    }
    // 내적/외적 구현
    double dot(const vector2& rhs) const {
        return x * rhs.x + y * rhs.y;
    }
    double cross(const vector2& rhs) const {
        retrun x * rhs.x + y * rhs.y;
    }
}
```

벡터 방향 판별

```
// 점 p를 기준으로 벡터 b가 벡터 a의 반시계 방향이면 양수, 시계 방향이면 음수, 평행이면 0을
반환
double ccw(vector2 p, vector2 a, vector2 b) {
    return ccw(a - p, b - p);
}
```

선분 교차

- 교차점이 필요한 경우

```
// (a, b), (c, d)가 평행한 두 선분일 때, 이들이 한 점에서 겹치는 지 확인
bool pararellSegments(vector2 a, vector2 b, vector2 c, vector2 d, vector2&
p) {
    if (b < a)
        swap(a, b);
    if (d < c)
```

```
        swap(c, d);
    // 한 직선 위에 없거나 두 선분이 겹치지 않는 경우를 우선 걸러낸다
    if (ccw(a, b, c) != 0 || b < c || d < a)
        return false;
    // 두 선분이 확실히 겹치므로 교차점을 찾는다
    if (a < c)
        p = c;
    else
        p = a;
    return true;
}

// p가 (a, b)를 감싸면서 각 변이 x, y축에 평행한 최소 사각형 내부에 있는 지 확인한다
// a, b, p는 일직선 상에 있다고 가정한다
bool inBoundingRectangle(vector2 p, vector2 a, vector2 b) {
    if (b < a)
        swap(a, b);
    return p == a || p == b || (a < p && p < b);
}

// (a, b)선분과 (c, d)선분의 교점을 p에 반환한다
// 교점이 여러 개일 경우 아무 점이나 반환
// 두 선분이 교차하지 않을 경우 false를 반환
bool segmentIntersection(vector2 a, vector2 b, vector2 c, vector2 d,
vector2& p) {
    // 두 직선이 평행인 경우를 우선 예외 처리한다
    if (!lineIntersection(a, b, c, d, p))
        return pararellSegments(a, b, c, d, p);
    return inBoundingRectangle(p, a, b) && inBoundingRectangle(p, c, d);
}
```

- 교차점이 필요 없을 때

```
// 두 선분이 서로 접촉하는 지 여부를 반환
bool segmentIntersects(vector2 a, vector2 b, vector2 c, vector2 d) {
    double ab = ccw(a, b, c) * ccw(a, b, d);
    double cd = ccw(c, d, a) * ccw(c, d, b);
    // 두 선분이 한 직선 위에 있거나 끝점이 겹치는 경우
    if (ab == 0 && cd == 0) {
        if (b < a)
            swap(a, b);
        if (d < c)
            swap(c, d);
        return !(b < c || d < a);
    }
    return ab <= 0 && cd <= 0;
}
```

점과 선 사이의 거리

```
// 점 p에서 (a, b) 직선에 내린 수선의 발을 구한다
vector2 perpendicularFoot(vector2 p, vector2 a, vector2 b) {
    return a + (p - a).project(b - a);
}
// 점 p와 직선(a, b)직선 사이의 거리를 구한다
double pointToLine(vector2 p, vector2 a, vector2 b) {
    return (p - perpendicularFoot(p, a, b).norm());
}
```

풀어보자

- GCD, https://www.acmicpc.net/problem/11689
- 잘못 구한 에라토스테네스의 체, https://www.acmicpc.net/problem/15897
- 서로소 그래프, https://www.acmicpc.net/problem/23832
- 선분 교차, https://www.acmicpc.net/problem/17387
- 선분과 점, https://www.acmicpc.net/problem/11664