#중급반 4주차

정수론과 조합론

<Number theory & Combinatorics>

#모듈로 연산

<Modulo Arithmetic> - 소개

알고리즘 문제를 풀다 보면, 정수론 문제들을 많이 볼 수 있습니다.
“정수론(Number Theory)”은 수학의 한 분야로, 정수의 성질을 연구합니다.
모듈로 연산은 정수끼리의 나눗셈에서 나머지를 구하는 연산으로,
정수론에서 많이 다뤄지는 연산 중 하나입니다.

정수론 내용은 방대해서 난이도 분포가 매우 다양하지만,
오늘은 모듈로 연산에서의 사칙연산에 대해서만 알아보겠습니다.

#모듈로 연산

<Modulo Arithmetic> - 소개

모듈로 연산에서의 사칙 연산은 다음과 같은 성질을 만족합니다.

1. 덧셈과 모듈로 연산의 순서를 바꾸어도 된다.
2. 곱셈과 모듈로 연산의 순서를 바꾸어도 된다.
3. 뺄셈과 모듈로 연산의 순서를 바꾸어도 된다.

덧셈을 예시로 들면, ((a%c)+(b%c))%c = (a+b)%c라는 것을 의미합니다.

증명은 a = pc+q, b = rc+s라고 가정하고 a, b에 식을 대입하면 됩니다.

#유클리드 호제법

<Euclidean algorithm> - 최대공약수

최대공약수 : 여러 수의 공통된 약수 중 가장 큰 수
C++에서 최대공약수를 어떻게 구할 수 있을까요?
다양한 방법으로 30과 42의 최대공약수를 구해봅시다!

1. Bruteforce를 이용한 최대공약수
2. 소인수분해를 이용한 최대공약수
3. 유클리드 호제법을 이용한 최대공약수

#유클리드 호제법

<Euclidean algorithm> - 최대공약수

1. Bruteforce

모든 경우의 수를 탐색하는 알고리즘입니다.

최대공약수는 원래 수보다는

작아야 한다는 성질이 있습니다.

따라서, 1부터 30까지만 수를 확인합니다.

1부터 30까지, 어떤 수 N의 값을 바꾸면서

30과 42를 N으로 나눈 나머지를 구합니다.

두 나머지의 값이 모두 0이 나오면

그 수는 공약수입니다.

시간복잡도는 O(N)입니다.

구현은 쉽지만 느리다는 단점이 있습니다.

```cpp
#include<iostream>

using namespace std;

int main() {

    int a = 30, b = 42;
    int res = -1;

    for (int i = 1; i < 30; i++) {
        if (!(a % i || b % i))res = i;
    }

    cout << res;
}
```

#유클리드 호제법

<Euclidean algorithm> - 최대공약수

2. 소인수분해

소인수분해를 이용하기 위해서는

일단 30과 42를 소인수분해해야 합니다.

30 = 2 × 3 × 5

42 = 2 × 3 × 7

따라서, 두 수의 최대공약수는 2 × 3이 됩니다.

하지만, C++로 소인수분해를 구현하는 것이 복잡하고,

시간복잡도 또한 $O(\log n + \sqrt{n})$으로 최적은 아닙니다.

#유클리드 호제법

<Euclidean algorithm> - 최대공약수

3. 유클리드 호제법

30과 42의 최대공약수는 6입니다.

42를 30으로 나눈 나머지는 12입니다.

이때, 12와 30의 최대공약수도 6이 나오게 됩니다.

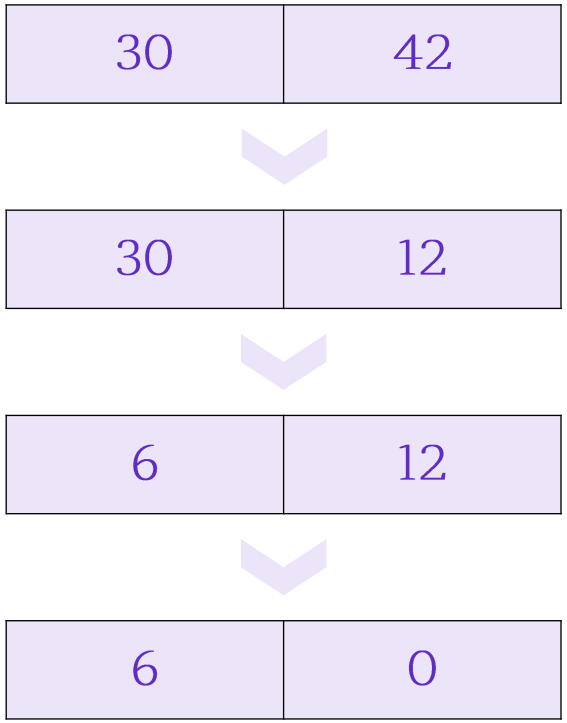
이와 같이, 두 수의 최대공약수는

큰 수에서 작은 수를 나눈 나머지와

작은 수의 최대공약수와 같습니다.

이를 통해 재귀적으로 최대공약수를 구할 수 있고,

이를 유클리드 호제법이라고 합니다.



#유클리드 호제법

<Euclidean algorithm> - 구현 코드

기본적으로 유클리드 호제법의 구현은
재귀함수를 통해 할 수 있습니다.
gcd(a, b)를 a, b의 최대공약수라고 하면,
b == 0일 때 a를 return하고
b != 0일 때 gcd(b, a%b)를 return합니다.
(단, a > b여야 합니다.)

시간복잡도는 O(logN)입니다.

```
int gcd(int a, int b) {
    if (a > b) return gcd(b, a);

    if (!b) return a;
    else return gcd(b, a % b);
}
```

#소수 판정

<Primality test> - 소개

소수 : 약수가 1과 자기 자신밖에 없는 1이 아닌 자연수
C++에서 어떤 수가 소수인지 아닌지를 어떻게 구할 수 있을까요?
다양한 방법으로 1000까지의 소수들을 구해봅시다!

1. Bruteforce를 이용한 소수 구하기
2. 더 빠른 Bruteforce
3. 에라토스테네스의 체

#소수 판정

<Primality test> - 구현 코드

1. Bruteforce

모든 경우의 수를 탐색하는 알고리즘입니다.

약수는 원래 수보다는

작거나 같아야 한다는 성질이 있습니다.

따라서, N이라는 수의 소수 판정을 할 때

1부터 N까지만 수를 확인합니다.

시간복잡도는 $O(N)$, N까지의 모든 수를 확인하려면 $O(N^2)$입니다.

```cpp
#include<iostream>

using namespace std;

const int max_number = 1000;

bool not_prime[max_number + 1];

void primality_test() {
    for (int i = 2; i < max_number + 1; i++) {
        for (int j = 2; j < i; j++) {
            if (!(i % j)) {
                not_prime[i] = true;
                break;
            }
        }
    }
}

int main() {

    primality_test();

    cout << boolalpha;

    cout << not_prime[12] << '\n'; // true(합성수)
    cout << not_prime[17] << '\n'; // false(소수)
}
```

#소수 판정

<Primality test> - 구현 코드

2. 빠른 Bruteforce

k가 N의 약수라면, $\frac{N}{k}$도 N의 약수입니다.

따라서, $\sqrt{N}$까지만 확인해주면 됩니다.

하나의 수에 대한 소수 여부를 확인할 때 주로 사용하는 방법입니다.

시간복잡도는 $O(\sqrt{N})$, N까지의 모든 수를 확인하려면 $O(N\sqrt{N})$입니다.

```cpp
#include<iostream>

using namespace std;

const int max_number = 1000;

bool not_prime[max_number + 1];

void primality_test() {
    for (int i = 2; i < max_number + 1; i++) {
        for (int j = 2; j * j <= i; j++) {
            if (!(i % j)) {
                not_prime[i] = true;
                break;
            }
        }
    }
}

int main() {

    primality_test();

    cout << boolalpha;

    cout << not_prime[12] << '\n'; // true(합성수)
    cout << not_prime[17] << '\n'; // false(소수)
}
```

#소수 판정

<Primality test> - 구현 코드

3. 에라토스테네스의 체

2부터 만약 현재 보고 있는 수가 소수라면

그 소수의 배수를 모두 지우는 방법입니다.

지워지지 않은 수는 모두 소수입니다.

소수를 한번에 판별해야 할 때 사용합니다.

시간복잡도는 $O(N \log(\log N))$입니다.

```cpp
#include<iostream>

using namespace std;

const int max_number = 1000;

bool not_prime[max_number + 1];

void primality_test() {
    for (int i = 2; i < max_number + 1; i++) {
        if (!not_prime[i]) {
            for (int j = 2*i; j <= max_number; j+=i) {
                not_prime[j] = true;
            }
        }
    }
}

int main() {

    primality_test();

    cout << boolalpha;

    cout << not_prime[12] << '\n'; // true(합성수)
    cout << not_prime[17] << '\n'; // false(소수)
}
```

#소수 판정

<Primality test> - 연습 문제

4 에라토스테네스의 체(BOJ #2960)

<문제 설명>

- N과 K가 주어진다.
- N까지의 수에 대해 에라토스테네스의 체를 생각한다.
- 이때, K번째 지워지는 수를 구하기

<제약 조건>

- $1 \leq K \leq N$
- $\max(1, K) < N \leq 1{,}000$

#소수 판정

<Primality test> - 구현 코드

<문제 해설>

지워진 수의 개수를 세는 변수를 만듭니다.

이후, 에라토스테네스의 체를 구현합니다.

지워지지 않은 수를 지우면

변수의 값을 1 늘립니다.

지워진 수의 개수가 K와 같아졌다면

그 때 지운 숫자가 답이 됩니다.

```
int main() {
    int N, K;
    cin >> N >> K;
    //지워졌나 판별하기 위한 vector 선언
    vector<bool> erase(N + 1);
    int cnt = 0;
    for (int i = 2; i <= N; i++) {
        for (int j = i; j <= N; j += i) {
            if (!erase[j]) {
                //새로운 수를 지웠을 때
                erase[j] = true;
                //cnt의 값 1 증가시킨다.
                cnt++;
            }
            if (cnt == K) {
                //만약 K개를 지웠다면
                //마지막으로 지운 수를 출력
                cout << j;
                //프로그램을 종료
                return 0;
            }
        }
    }
}
```

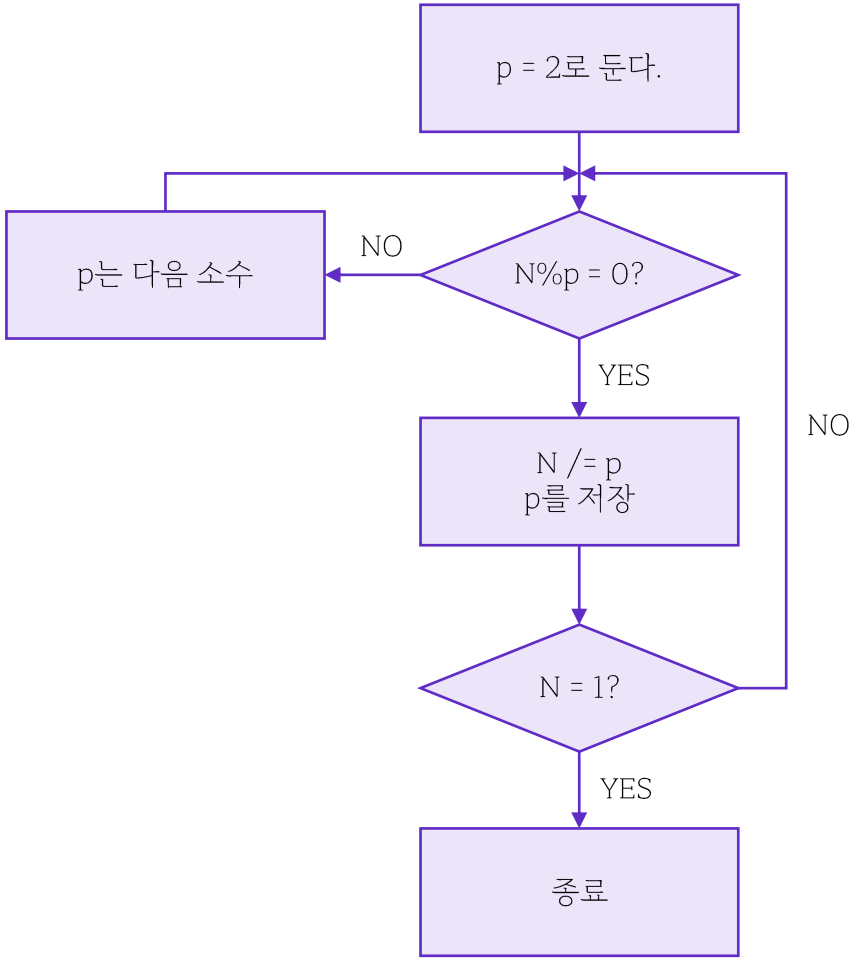
#소인수분해

<Prime factorization> - 소개

소인수분해 : 어떤 수를 소인수의 곱으로 표현하는 것

먼저, 순서도를 간략하게 그려보겠습니다.
순서도를 간단히 요약하면, N을 가장 작은 소수부터 나눠보는 방법입니다.
하지만, 이 알고리즘은 O(N)으로, 최적은 아닙니다.

p가 N의 약수라면, $\frac{N}{p}$ 또한 N의 약수라는 사실을 통해,
나눠볼 수를 줄일 수 있습니다.



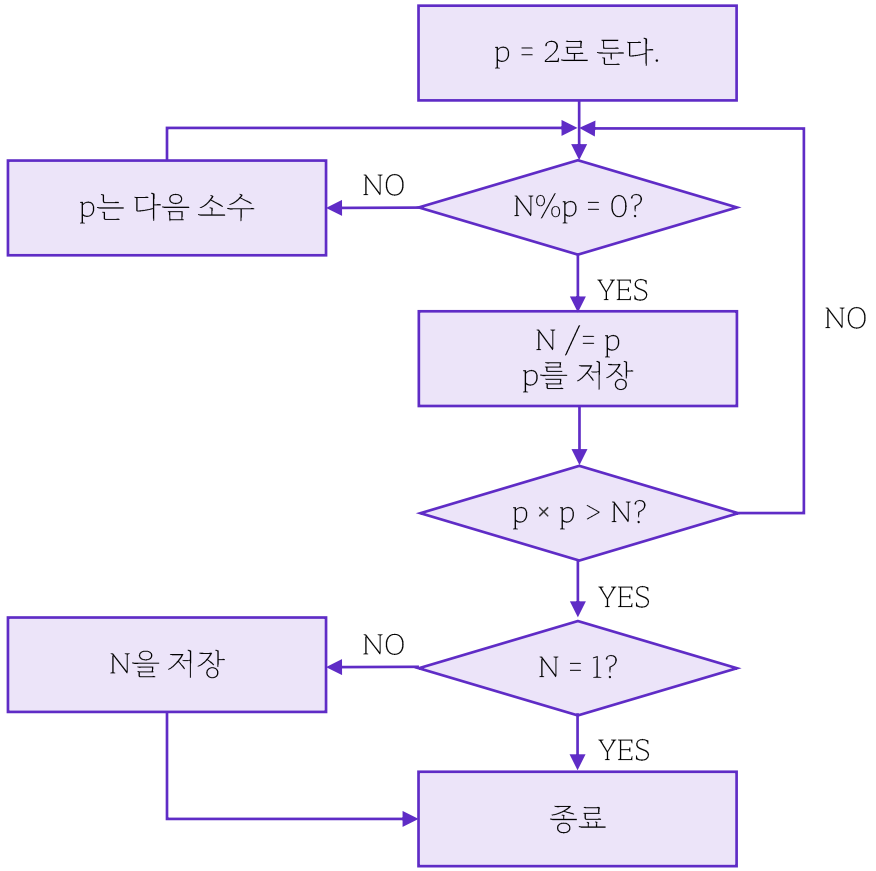
#소인수분해

<Prime factorization> - 소개

소수 판정 방법 중 두 번째 방법을 기억하시나요?

Bruteforce라도, $\sqrt{N}$까지만 확인을 해서 소수임을 판별할 수 있었습니다.
소인수분해도 $\sqrt{N}$까지만 확인을 했는데도 만약 N이 1이 되지 않았다면,
N은 소수이고, 나머지 숫자들을 굳이 확인하지 않아도
N을 그대로 적어두면 소인수분해가 끝난다는 것을 알 수 있습니다.
따라서, 시간복잡도는 $O(\sqrt{N})$입니다.

이제 오른쪽에 있는 순서도를 코드로 옮겨보겠습니다.



#소인수분해

<Prime factorization> - 구현 코드

오른쪽 함수는 소인수분해를 하는 함수입니다.

vector<pair<int, int>>에서

first는 소인수, second는 개수입니다.

코드에서는 직접 다음 소수로 넘어가는 것이 힘들기 때문에, p를 1씩 늘리며 소수인 경우만 체크하는 방식으로 작성했습니다.

에라토스테네스의 체는 생략했습니다.

실제로 not_prime 배열이 전처리 되어있어야 합니다.

오른쪽에서 not_prime은 소수인 경우 false,

나머지 경우에 true가 저장되어 있습니다.

```
void prime_factorization(int n) {
    int p = 2;
    vector<pair<int, int>> v;
    while (n != 1) {
        if (!not_prime[p]) {
            if (!(n % p)) {
                int tmp = 0;
                while (!(n % p)) {
                    tmp++;
                    n /= p;
                }
                v.push_back(make_pair(p, tmp));
            }
        }
        p++;
        if (p * p > n && n != 1) {
            v.push_back(make_pair(n, 1));
            break;
        }
    }
    for (auto pii : v) {
        cout << pii.first << ' ' << pii.second << '\n';
    }
}
```

#조합론

<Combinatorics> - 소개

"조합론(Combinatorics)"이란, 경우의 수에 대해 다루는 수학의 한 분야입니다.
조합론에서는 다음과 같은 내용을 다룰 예정입니다.

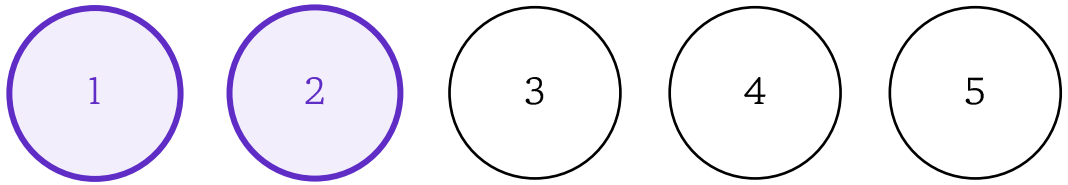
조합론 역시 정수론과 마찬가지로 양이 꽤 많습니다.
하지만, 그만큼 문제도 다양하기 때문에 solved.ac 티어를 올리기 좋습니다.

1. 순열과 조합의 정의
2. C++에서 C(n, k)의 값 구하기
3. 합의 법칙과 곱의 법칙

#조합론

<Combinatorics> - 순열과 조합

1. 순열과 조합의 정의



<조합>

Elements	Count
1, 2	1
1, 3	2
…	…
4, 5	10

<순열>

Elements	Count
1, 2	1
2, 1	2
…	…
5, 4	20

조합 : n개 중 k개를 뽑는 경우의 수

순열 : n개 중 k개를 뽑고 나열하는 경우의 수

#조합론

<Combinatorics> - 순열과 조합

1. 순열과 조합의 정의

n개 중 k개를 뽑을 때,
조합은 C(n, k)로, 순열은 P(n, k)로 나타냅니다.
각각을 구하는 식은 다음과 같습니다.

$$C(n,k) = \frac{n!}{k!\,(n-k)!}, \qquad P(n,k) = \frac{n!}{(n-k)!}$$

따라서, 다음과 같은 성질을 만족합니다.

$$C(n,k) \times k! = P(n,k)$$

참고로, C(n, 0) = C(n, n) = 1 입니다.

#조합론

<Combinatorics> - 조합 구현하기

2. C++에서 C(n, k)의 값 구하기

그렇다면, 이제 C(n, k)를 구하는 코드를 구현해봅시다.
구현에는 다음과 같은 방법이 있습니다.

1. 직접 계산하기
2. 팩토리얼만 계산해놓기
3. 파스칼의 삼각형 이용하기

#조합론

<Combinatorics> - 조합 구현하기

2 – 1 . 직접 계산하기

Combination의 정의에 따라
수를 직접 계산해줍니다.

시간복잡도는 O(N)이 됩니다.

원래 Combination의 계산 결과는 크게 나오기 때문에
특정한 수로 나눈 나머지를 물어봅니다.
이를 위해서는 모듈러 연산에서의 나눗셈을 알아야 하는데,
아직은 어려운 내용이므로 생략했습니다.

```
int Combiation(int n, int k) {
    int ret = 1;
    for (int i = n - k + 1; i <= n; i++) {
        ret *= i;
    }
    for (int i = 1; i <= k; i++) {
        ret /= i;
    }
    return ret;
}
```

#조합론

<Combinatorics> - 조합 구현하기

2 – 2 . 팩토리얼만 계산해놓기

Combination을 구할 때
팩토리얼을 많이 사용하게 됩니다.
따라서, Combination을 여러 번 계산할 때
팩토리얼만 계산해 놓으면
빠르게 계산을 할 수 있습니다.

준비하기 위한 시간복잡도는 O(N),
계산하는 데 필요한 시간복잡도는 O(1)입니다.

```
const int MAX_NUMBER = 10;

int factorial[MAX_NUMBER + 1];

void init() {
    factorial[0] = 1;
    for (int i = 1; i <= MAX_NUMBER; i++) {
        factorial[i] = factorial[i - 1] * i;
    }
}

int Combiation(int n, int k){
    return factorial[n] / (factorial[n - k] * factorial[k]);
}
```

#조합론

<Combinatorics> - 조합 구현하기

2 – 3 . 파스칼의 삼각형 이용하기

Combination의 중요한 성질이 있습니다.
C(n, k) + C(n, k + 1) = C(n + 1, k + 1)이 됩니다.
따라서, DP처럼 접근을 해볼 수도 있습니다.

합을 이용하여 Combination을 구하기 때문에
나눗셈을 안해도 됩니다.
즉, Combination을 어떤 수로 나눈
나머지를 계산할 때 사용할 수 있습니다.

시간복잡도는 $O(N^2)$입니다.

```
const int MAX_NUMBER = 10;

int combination[MAX_NUMBER + 1][MAX_NUMBER + 1];          memo할 배열을 만듭니다.

void init() {
    for (int i = 1; i <= MAX_NUMBER; i++) {
        for (int j = 0; j <= i; j++) {
            if (!j || j == i) combination[i][j] = 1;      C(n, 0) = C(n, n) = 1 입니다.
            else combination[i][j] = combination[i - 1][j - 1] +
combination[i - 1][j];
        }
    }
}
```

#조합론

<Combinatorics> - 합의 법칙과 곱의 법칙

3. 합의 법칙과 곱의 법칙

두 경우의 수를 합쳐야 할 때, 합의 법칙 또는 곱의 법칙을 사용할 수 있습니다.

이름 그대로, 합의 법칙은 두 경우의 수를 합하는 경우, 곱의 법칙은 두 경우의 수를 곱하는 경우입니다.

합의 법칙은 두 사건 중 하나만 일어나도 상관이 없는 경우, 별개의 두 경우일 때 사용하고
곱의 법칙은 두 사건이 모두 일어나야 하는 경우, 동시에 일어나는 경우일 때 사용합니다.

#조합론

<Combinatorics> - 연습문제

이항 계수 1(BOJ #11050)

<문제 설명>

- N과 K가 주어진다.
- 이때, C(N, K)의 값을 구하기

<제약 조건>

- $1 \leq N \leq 10$
- $0 \leq K \leq N$

#조합론

<Combinatorics> - 연습문제

<문제 해설>

이 문제는 제한이 작기 때문에

팩토리얼로 이항 계수를 직접 구하면 됩니다.

코드는 오른쪽을 참고하시면 됩니다.

```
int Combiation(int n, int k) {
    int ret = 1;
    for (int i = n - k + 1; i <= n; i++) {
        ret *= i;
    }
    for (int i = 1; i <= k; i++) {
        ret /= i;
    }
    return ret;
}
```

#조합론

<Combinatorics> - 연습문제

이항 계수 2(BOJ #11051)

<문제 설명>

- N과 K가 주어진다.
- 이때, C(N, K)를 10,007로 나눈 값을 구하기

<제약 조건>

- $1 \leq N \leq 1{,}000$
- $0 \leq K \leq N$

#조합론

<Combinatorics> - 연습문제

<문제 해설>

제한이 큰 대신 나머지를 구하는 문제입니다.

따라서, 파스칼의 삼각형을 이용하면 됩니다.

정수론 부분에서 이야기한 성질인

"사칙연산과 모듈로 연산을 교환해도 된다"

라는 성질을 이용할 것입니다.

코드는 오른쪽과 같습니다.

```
const int MAX_NUMBER = 1000;
const int MOD = 10007;

int combination[MAX_NUMBER + 1][MAX_NUMBER + 1];

void init() {
    for (int i = 1; i <= MAX_NUMBER; i++) {
        for (int j = 0; j <= i; j++) {
            if (!j || j == i) combination[i][j] = 1;
            else combination[i][j] = (combination[i - 1][j - 1] +
combination[i - 1][j]) % MOD;                    더한 후 10007로 모듈로 연산을 해줍니다.
        }
    }
}
```

#연습 문제 도전

정수론

5 아주 간단한 문제 (BOJ #25375)

아주 간단한 문제

2 골드바흐 파티션 (BOJ #17103)

제한이 늘어나면 다이아가 되는 신기한 문제

2 Messi An-Gimossi (BOJ #17355)

GOAT

4 일어나... 코딩해야지... (BOJ #24462)

HCPC 기출문제

5 라그랑주님 수학에는 뺄셈도 있어요 (BOJ #25399)

정수론에는 이런 문제도 있어요!

4 GCD 곱 (BOJ #14860)

어렵지만 한번 도전해보세요

#연습 문제 도전

조합론

5 다각형의 대각선 (BOJ #3049)

Well-Known

5 암호 (BOJ #1394)

하나하나 해보시면 시간초과가 납니다!

2 N포커 (BOJ #16565)

아이패드/종이와 함께하면 쉬워져요

2 방향 정하기 (BOJ #26524)

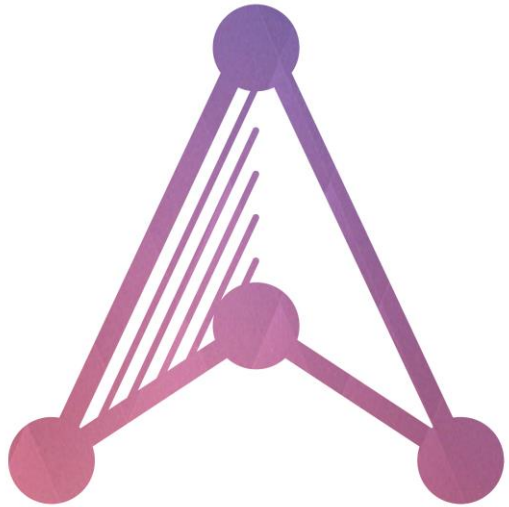
생각보다 간단한 문제일수도?

4 합분해 2 (BOJ #13707)

Combination의 응용

5 고층 빌딩 (BOJ #1328)

수학을 좋아하는 저의 첫 플래티넘 문제 / DP + 조합론



ALOHA

The algorithm club.