\*\* int쓸 때 long long int 생각하기!!

\*\*실수형쓸 때 오차생기므로 string으로 바꾸든, slicing을 하든 처리해준뒤 써야한다

s : 저장할배열, arr : 원소들어있는배열, n : 필요한원소갯수, ind : s인덱스, k: arr인덱스

//부분집합 출력 0, 1, 2, 3 ->

0123 012 013 1 023 02 03 0 123 12 13 1 23 2 3 이렇게 오름차순으로 출력됨

void func(int\* s, int\* arr, int n, int k, int ind) {

if (n == k) {

print(s); return;

}

s[ind] = arr[k];

func(s, arr, n, k + 1, ind + 1);

func(s, arr, n, k + 1, ind + 1);

}

//조합출력 0,1,2,3중 2개 -> 01 02 03 12 13 23

void func(int\* s, int\* arr, int n, int k, int ind) {

if (n == 0) {

for (int i = 0; i < 2; i++) {

cout << s[i] << " ";

}

cout << endl;

return;

}

if (k == 4) { return; }

s[ind] = arr[k];

func(s, arr, n-1, k + 1, ind + 1); //s의 원소갯수 1감소, 다음원소 인덱스 증가, s인덱스 증가

func(s, arr, n, k + 1, ind); //s원소갯수 그대로, 다음원소 인덱스증가, s인덱스 그대로

}

//중복조합출력 0, 1, 2, 3 중 2개 -> 00 01 02 03 11 12 13 22 23 33

void func(int\* s, int\* arr, int n, int k, int ind) {

if (n == 0) {

for (int i = 0; i < 2; i++) {

cout << s[i] << " ";

}

cout << endl;

return;

}

else if (k == 4) { return; }

else {

s[ind] = arr[k];

func(s, arr, n - 1, k, ind + 1);

func(s, arr, n, k + 1, ind);

}

}

//중복순열 0, 1, 2 중 2개 -> 00 01 02 10 11 12 20 21 22

void func(int\* s, int\* arr, int k, int n, int ind) {

if (k == 0) {

for (int i = 0; i < 2; i++) {

cout << s[i] << " ";

}

cout << endl;

return;

}

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

s[ind] = arr[i];

func(s, arr, k - 1, n, ind + 1);

}

return;

}

//순열출력 0,1,2,3중 2개 01 02 03 10 12 13 20 21 23 30 31 32

Void func(arr, ind, k){

If(ind == k){print(arr,ind); return;}

For(i=ind; i<arr.length(); i++){

Right\_rotate(arr, ind, i);

Func(arr, ind+1, k);

Left\_rotate(arr, ind, i);

}

Ex)복면산문제 -> 순열출력 이용

Int check(int\* arr){

If(arr[0] == 0){return 0;} //s가 0이면 안됨

If(arr[4]==0){return 0;} //m이 0이면 안됨

If(1000\*arr[0]+ …==10000\*arr[4]+…){return 1;}

Return 0;

}

Void func(arr, len, k){

If(len == k){

If(check(arr)){print(arr, len); return;}

For(i=len; i<arr.length(); i++){

Swap\_arr(arr, len, i);

Func(arr, len+1, k);

Swap\_arr(arr, len, i);

}

}

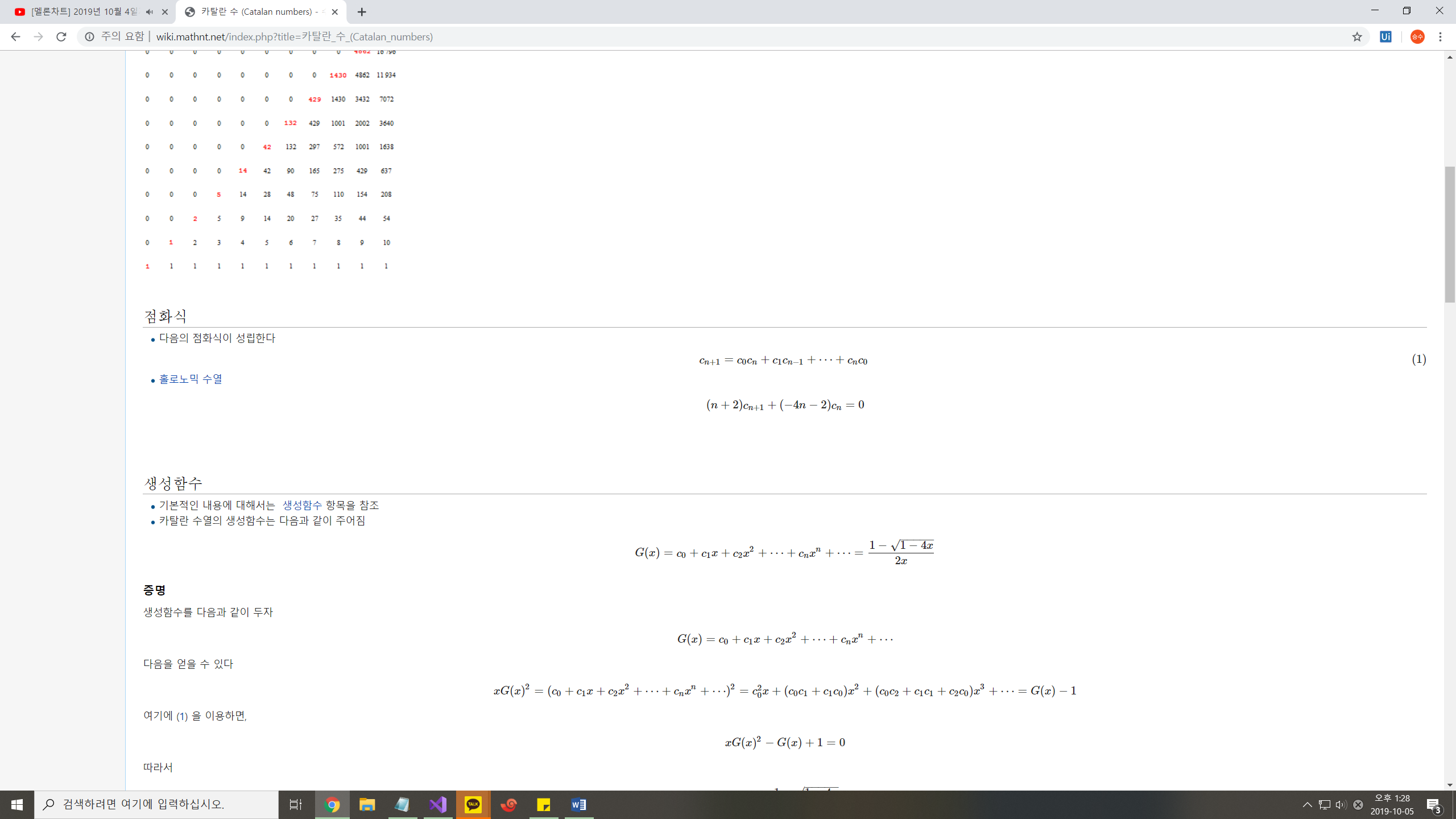
//nCr

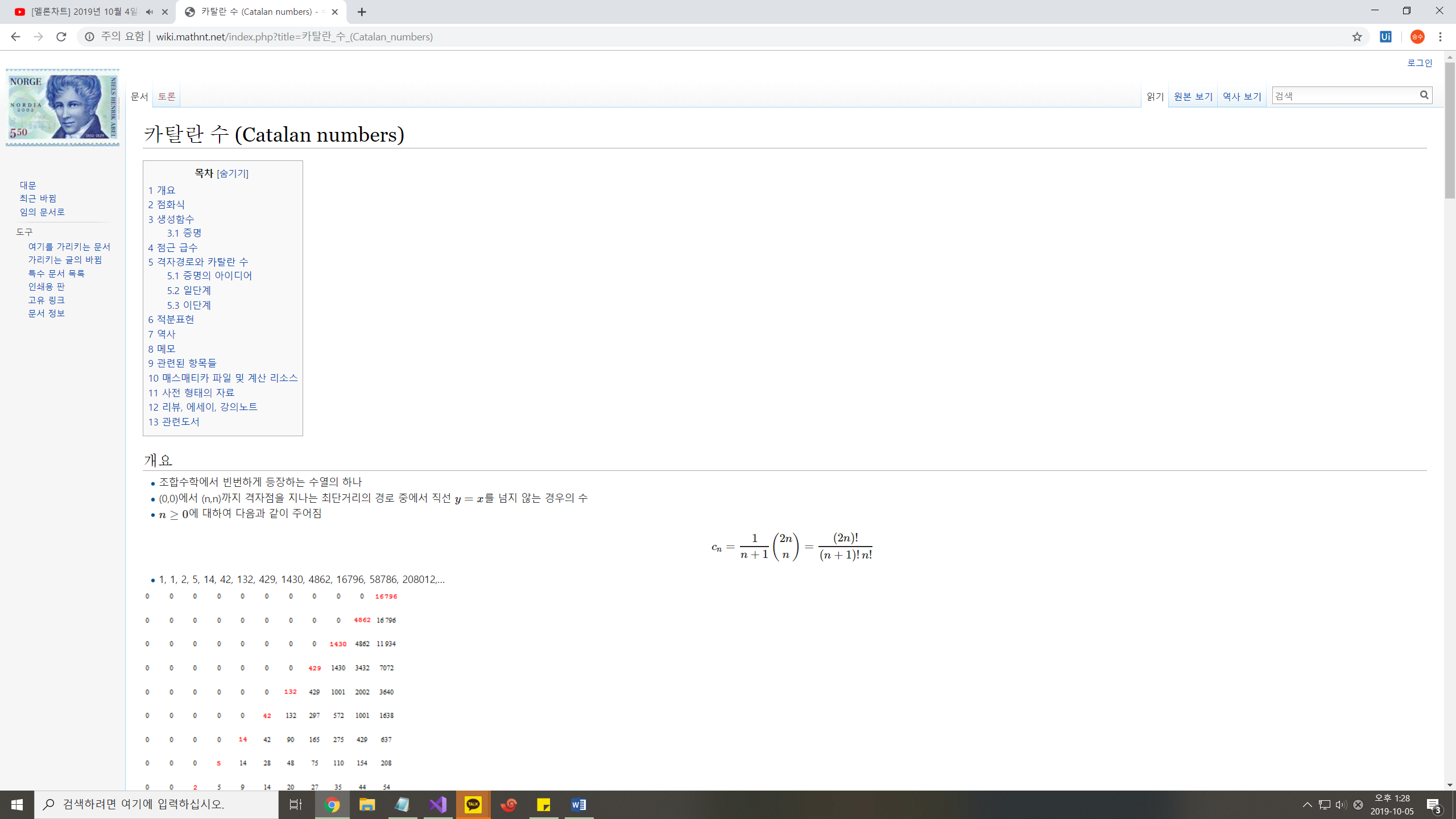
nCr = n-1Cr-1 + n-1Cr;

nCr에서 n==r이거나 r=0이면 nCr = 1

For(i=0; i<=n; i++){sum+=nCi}하면 sum은 2^n.

//카탈란수





 1, 1, 2, 5, 14, 42, 132, 429

-쌍 있는 괄호조합갯수

-원 위의 2n개의 점 중 어느 두 현도 만나지않도록 두점씩 잇는방법수

-격자점 지내는 최단거리 경로의 수(0,0) ~ (n,n)

-이진트리 만들수있는 종류 수

//랜덤함수

#include <stdlib.h>

#include <time.h>

srand(time(0));

Num = rand() % 100

Num2 = (double)(rand()%100)/RAND\_MAX;

//님문제 xor로

각접시 남은 동전수를 a, b, c, d라하면 택(a,b,c,d)의 결과가0이면 안정적

상대한테 안정적인경우를 계속 주면된다.

//정렬

#include <algorithm>

Bool compare(int a, int b){

Return a<b;

}

Int\* arr = new int[10];

sort(arr, arr+sizeof(arr),compare);

//벡터사용법

#include<vector>

Vector<int> a(100);

Sort(a.begin(),a.end());

마찬가지로

Sort(a.begin(), a.end(),compare);

//그레이코드

void func(int\* arr, int n, int ind, int reverse) {

if (n == ind) { print(arr); return; }

arr[ind] = reverse;

func(arr, n, ind + 1, 0);

arr[ind] = 1 - reverse;

func(arr, n, ind + 1, 1);

return;

}

//연결리스트출력

void func(node\* from) {

if (from == NULL) { return; }

cout << from.value << endl;

func(from->next);

}

//역순으로 출력은 순서만 바꾸면된다

void func(node\* from) {

if (from == NULL) { return; }

func(from->next);

cout << from.value << endl;

}//올림(ceil) 내림(floor)

#include <cmath>

반올림은 올리고싶은자리\*(1/2)한다음 floor하면된다

Ex) 4567 백의자리에서 반올림 -> 50을더하고 floor 즉 4617을 백의자리내림 = 4600

\*올림, 내림은 수직선상으로본다. 즉 -4.2올림은 -4, 내림은 -5

//소수판별 –cmath에 sqrt(num)있다

2~sqrt(n)까지 나눠보기 -> 에라토스테네스방법 ->소수판별

2부터 n까지 나누는데 맨 처음수만 소수고 그 수의 배수를 모두 지운다 ->n까지의 모든소수 find

-> 에라토스테네스의채

//최대공약수 ->유클리드호제법

int func(int a, int b) {

if (a < b) { return func(b, a); }

if (a % b == 0) { return b; }

return func(b, a % b);

}

//구간합 -> i~j인덱스까지의 합

누적합을 구한뒤 sum[j]-sum[i] = i~j까지의 합

//금액지불하는 방법 수 1,2,5,10,20,50만원으로 100만원

1,2,5,10,20,으로 100만원 + (1,2,5,10,20으로 50만원 + 50만원 1개) + (1,2,5,10,20으로 0만원 + 50만원2개)

Pay(m, n) = m종류의 연속된 지폐 이용해 n원 지불

Pay(m, n) = pay(m-1, n-bills[m-1]\*i) i는 0부터 n/bills[m-1]까지다

Pay(1,n)에서 n이 bills[0]으로 나눠지면 1, 안나눠지면 0

int func(int\* bills, int m, int n) {

if (m == 1) {

if (n % bills[0] == 0) { return 1; }

else { return 0; }

}

int count = 0;

for (int i = 0; i <= n / bills[m - 1]; i++) {

count += func(bills, m - 1, n - bills[m - 1] \* i);

}

return count;

}

//수분할

Div(m,n) -> m을 n이하의 자연수로 표현

Div(5,3) = 3으로 시작하는경우 -> div(2,2)

2로 시작하는 경우 -> div(3,2)

1로 시작하는 경우 -> div(4,1)

Div(m, n) = if(m != 0 && m<n)이면 n=m

Div(m, n) = div(m-I, i) i는 1에서 n까지 합

Dib(0,n) = 1

int func(int m, int n) {

int sum = 0;

if (n > m) { n = m; }

if (m == 0) { return 1; }

else {

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

sum += func(m - i, i);

}

return sum;

}

}근데 중복계산 때문에 메모이제이션쓴다 -> arr[m][m]으로

//순서고려한 수분할 4->1+1+1+1/ 1+1+2/ 1+2+1/ 1+3/ 2+1+1/ 2+2/ 3+1/ 4 -> 8개

즉 n분할 -> 2^n-1개

출력은 배열에 넣고 한 개씩 출력

void func(int\* arr, int ind, int n) {

if (n == 0) {

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

cout << arr[i] << " ";

};

cout << endl;

return;

}

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

arr[ind] = i;

func(arr, ind + 1, n - i);

}

}

//구조체선언

struct Employee { short id*;* int age*;* float wage*;* }*;*

struct Company {

Employee CEO

*struct* int numberOfEmployees*;*

}*;*

 Company myCompany = {{ 1, 42, 60000.0f }, 5 }*;*