Number Theory

정수론 기초

Dosawas 2024-05-11

목차

차례

- 5회차 리뷰
- 소수 판정 : 에라토스테네스의 체, 소인수분해(factorization)
- GCD 구하기 : 확장 유클리드 호제법
- 이항계수 :
  - DP로 이항계수 구하기
  - 모듈러역원을 이용한 이항계수 구하기
- 깃허브에 자기가 자주 쓰는 도구 저장하기

5회차 리뷰

- DP는 많은 문제를 풀어서 점화식을 어떻게 짜는지 감을 잡는게 좋습니다.
- 배낭문제
- LIS
- LCS
- 구간 DP

- 각 유형이 뭐였는지 느낌만 잡아도 다음에 볼 때 참고해서 풀 수 있을 것입니다.

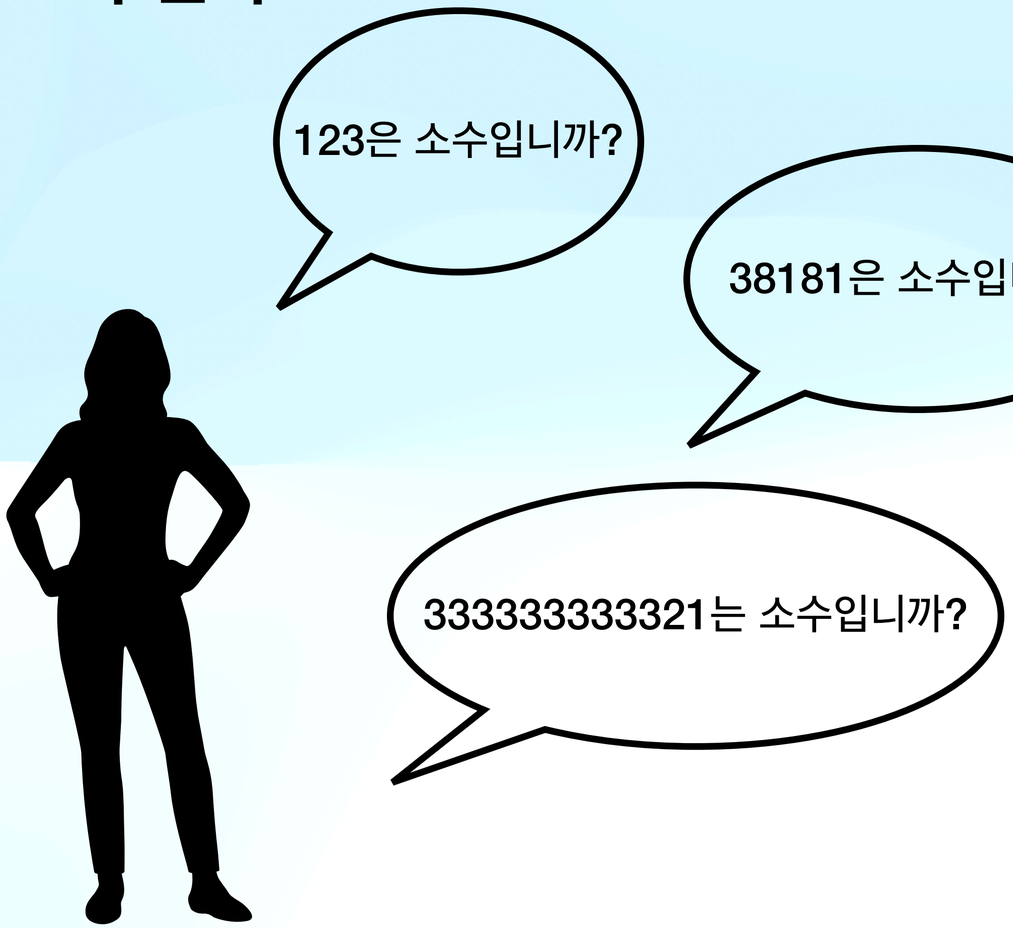
소수판정

이 수는 소수인가요?

- 몇초안에 대답했나요?
- 1은 소수가 아닙니다.
- 컴퓨터가 소수를 어떻게 구해야할지 생각해봅시다.
  - 자기보다 작은 수로 나눠본다.

소수판정

이 수는 소수인가요?



- 대답할 수 있나요?
- 333333333321의 소수인지 아닌지 어떻게 알 수 있을까요?

소수판정

이 수는 소수인가요?

- 사실 여기 있는 모든 수는 3으로 나누어떨어져서 소수가 아닙니다.
  - 컴퓨터로 break를 하면 금방 될 겁니다.

- 524287 은 소수입니다.
- 위의 논리로 계산하면
  - 숫자만큼의 시간이 걸릴겁니다.

소수판정

이 구간에 소수가 어떤게 있나요?

- 그냥 숫자 하나만 있으면 어떻게든 계산해보겠지만..
- 1부터 500000 사이에 모든 소수를 찾아주세요!!! 라고 하면 어떻게 할까요?
- IsPrime : $O(x)$
- find_all_prime $O(x \times n)$
- n이 10000만 넘어가도 ㅠㅠ

```
12    bool isPrime(int x){ // x가 소수인가?
13        if(x == 1) return false;
14        for(int i=2;i<x;i++) if(x % i == 0) return false;
15        return true;
16    }
17
18    vector<int> find_all_prime(int n){ // 1~n까지의 모든 소수를 리턴
19        vector<int> ret;
20        for(int i=1;i<=n;i++) if(isPrime(i)) ret.push_back(i);
21        return ret;
22    }
```

소수판정 : 에라토스테네스의 체

2960 에라토스테네스의 체

- 이 문제는 에라토스테네스의 체에 대해 설명합니다.

- 왜 체인지 그림으로 생각해봅시다.

1. 2부터 N까지 모든 정수를 적는다.
2. 아직 지우지 않은 수 중 가장 작은 수 P를 찾는다. (소수 확정)
3. P를 지우고 P의 배수를 지운다. (합성수 확정)
4. 아직 모든 수를 지우지 않았다면, 다시 2번으로 간다.

에라토스테네스의 체 성공 다국어

실버 IV

시간 제한	메모리 제한	제출
1 초	128 MB	28234

문제

에라토스테네스의 체는 N보다 작거나 같은 모든 소수를 찾는 유명한 알고리즘이다.

이 알고리즘은 다음과 같다.

1. 2부터 N까지 모든 정수를 적는다.
2. 아직 지우지 않은 수 중 가장 작은 수를 찾는다. 이것을 P라고 하고, 이 수는 소수이다.
3. P를 지우고, 아직 지우지 않은 P의 배수를 크기 순서대로 지운다.
4. 아직 모든 수를 지우지 않았다면, 다시 2번 단계로 간다.

N, K가 주어졌을 때, K번째 지우는 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N과 K가 주어진다. ($1 \le K < N$, $\max(1, K) < N \le 1000$)

소수판정 : 에라토스테네스의 체

2960 에라토스테네스의 체

- 1은 소수가 아니니까 빨간색으로 지웁시다
- 그럼 처음으로 안지운 수 2는 소수입니다.
  - 파란색으로 색칠하고
  - 그의 배수들은 빨간색으로 칠해봅시다.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

1. 2부터 N까지 모든 정수를 적는다.
2. 아직 지우지 않은 수 중 가장 작은 수 P를 찾는다. (소수 확정)
3. P를 지우고 P의 배수를 지운다. (합성수 확정)
4. 아직 모든 수를 지우지 않았다면, 다시 2번으로 간다.

소수판정 : 에라토스테네스의 체

2960 에라토스테네스의 체

- 그 다음 안지워진 수는 3이네요
  - 3을 파란색으로 칠하고
  - 3의 배수를 빨간색으로 칠해봅시다.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

1. 2부터 N까지 모든 정수를 적는다.
2. 아직 지우지 않은 수 중 가장 작은 수 P를 찾는다. (소수 확정)
3. P를 지우고 P의 배수를 지운다. (합성수 확정)
4. 아직 모든 수를 지우지 않았다면, 다시 2번으로 간다.

소수판정 : 에라토스테네스의 체

2960 에라토스테네스의 체

- 이제 끝까지 해봅시다.

- 1부터 100까지 수 중에서 소수를 잘 골라냈습니다.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

1. 2부터 N까지 모든 정수를 적는다.
2. 아직 지우지 않은 수 중 가장 작은 수 P를 찾는다. (소수 확정)
3. P를 지우고 P의 배수를 지운다. (합성수 확정)
4. 아직 모든 수를 지우지 않았다면, 다시 2번으로 간다.

소수판정 : 에라토스테네스의 체

2960 에라토스테네스의 체

```
12  vector<ll> eratos(int n){ // n은 1이상의 자연수
13      vector<ll> table(n+1, 0), primes;
14      // table[i] = i면 소수
15      // table[i] != i면 합성수고 i를 인수로 가짐
16      for(ll i = 2 ; i <= n; i++){
17          if(table[i]) continue;
18          table[i] = i; // 색칠하고
19          primes.push_back(table[i]);
20          for(ll j = i * i; j < n; j += i){
21              table[i] = i;
22          }
23      }
24      return primes;
25      // return table;
26  }
```

소수면 그냥 그 소수가 저장되어있고

합성수면 인수로 가지는 소수가 저장되어있다.

-> 이건 나중에 소인수분해에 쓰인다.

<- 여기를 보면 오버플로우를 대비해

Long long 형을 이용했습니다.

왜 i * i부터 계산해도 되는지 생각해봅시다.

소인수분해

- 12을 소인수분해 해볼까요?
  - 1 * 12
  - 2 * 6
  - 3 * 4
  - 그러니까 1, 2, 3, 4, 6, 12 !!
- 이런 생각을 컴퓨터로 나타내면
  코드는 다음과 같습니다.

```
61    vector<ll> factors(int x){ // x를 소인수분해
62        vector<ll> f;
63        for(ll i = 1; i * i<= x; i++){
64            if(x % i == 0){
65                ll a = i;
66                ll b = x / i;
67                if(a != b) f.push_back(a);
68                f.push_back(b);
69            }
70        }
71        return f;
72    }
```

- $O(\sqrt{x})$의 시간 복잡도를 가질 것 입니다.

소인수분해

- 그럼 이제 1부터 n까지 모든 수를 소인수분해 해볼까요?
- 아까 그 코드를 쓴다면 $O(n\sqrt{n})$의 시간복잡도가 걸릴것입니다.
- 더 좋은 방법은 없을까요?

소인수분해 - 쿼리

16563 어려운 소인수분해

문제

지원이는 대회에 출제할 문제에 대해서 고민하다가 소인수분해 문제를 출제해야겠다고 마음을 먹었다. 그러나 그 이야기를 들은 동생의 반응은 지원이의 기분을 상하게 했다.

> "소인수분해? 그거 너무 쉬운 거 아니야?"

지원이는 소인수분해의 어려움을 알려주고자 엄청난 자신감을 가진 동생에게 2와 500만 사이의 자연수 N개를 주고 소인수분해를 시켰다. 그러자 지원이의 동생은 기겁하며 쓰러졌다.

힘들어하는 지원이의 동생을 대신해서 여러분이 이것도 쉽다는 것을 보여주자!

입력

첫째 줄에는 자연수의 개수 N ($1 \le N \le 1,000,000$)이 주어진다.

둘째 줄에는 자연수 k_i ($2 \le k_i \le 5,000,000$, $1 \le i \le N$)가 N개 주어진다.

- 만약 시간복잡도 $O(n\sqrt{n})$에 풀면 시간초과를 받을 것입니다.
- 에라토스테네스의 체를 왜 아까 그런식으로 정의했는지 이제 나옵니다.

소인수분해

```cpp
const int MAX = 5e6;
NumTheory<MAX> nt; // table 용량땜에 전역변수로 했음
int main(){
    fast_io
    int n; cin >> n;
    nt.eratos();
    for(int i=0;i<n;i++){
        int k; cin >> k;
        vector<int> factors = nt.factorization(k);
        for(int f : factors){
            cout << f << ' ';
        }
        cout << '\n';
    }
}
```

시간복잡도 : $O(N \log_2 N)$보다 빠름

소인수가 최대 몇개정도 있을까를 생각해보자

```cpp
template <int SZ>
struct NumTheory{
    vector<ll> primes;
    ll table[SZ + 1];
    vector<int> eratos(int sz = SZ){
        table[1] = 1; // 소인수분해 할때만 1로 해둡시다.예외로
        for(ll i = 2;i<sz;i++){
            if(table[i]) continue;
            table[i] = i;
            primes.push_back(i);
            for(ll j = i * i; j < sz ; j += i){
                table[j] = i;
            }
        }
        return primes;
    }
    vector<int> factorization(int n){
        vector<int> ret;
        while(n != 1){
            ret.push_back(table[n]);
            n /= table[n];
        }
        // ret.push_back(1);
        sort(ret.begin(), ret.end());
        return ret;
    }
};
```

GCD 구하기

- GCD : Greatest Common Divisor - 최대공약수

- Naive한 방법 : gcd를 구할 두 수 중에서 작은 수까지 모든 수를 나눠보며 나눠지는 가장 큰 수를 찾는다.

- 하지만, gcd(1234567890, 8372613012)와 같이 큰 수를 계산하기는 힘들것입니다.

GCD 구하기 - 유클리드 알고리즘

```cpp
ll gcd(ll a, ll b){
    return (b ? a : gcd(b, a % b));
}
```

- 옆의 코드가 동작하는 방식을 봅시다.
- EX) gcd(396, 666)

$$396 \div 666 = 0 \ldots 396$$

$$666 \div 396 = 1 \ldots 270$$

$$396 \div 270 = 1 \ldots 126$$

$$270 \div 126 = 2 \ldots 18$$

$$126 \div 18 = 7 \ldots 0$$

결론 : gcd(396, 666) = 18이구나~

```
print(666 % 396)
print(396 % 270)
print(270 % 126)
print(270 % 126)
print(126 % 18)
```

[5] ✓ 0.0s

```
270
126
18
0
```

GCD 구하기 - 유클리드 알고리즘

```
ll gcd(ll a, ll b){
    return (b ? a : gcd(b, a % b));
}
```

- 옆의 코드가 왜 동작하는지 봅시다.
- $G = gcd(a, b)$
  $a = a'G, b = b'G$ 여기서 a 프라임과 b 프라임은 서로소겠지요.
- $a = bq + r \rightarrow a'G = b'Gq + r \rightarrow r = (a' - b'q)G$
  - **마지막 부분만 보면 나머지 r은 G를 약수로 가지고 있습니다!!**
  - 이제, $\gcd(a, b)$를 $\gcd(b, r)$로 바꾸려면 b'와 $a' - b'q$가 서로소여야합니다.
    - 귀류법을 쓰면, $b' = np, a' - b'q = mp, a' - npq = mp$
    - $a' = p(m + nq), b' = np$이므로 모순!! 따라서 $b', a' - b'q$는 서로소이다.
  - **결론 :** $\gcd(a, b) = gcd(b, r)$

확장 유클리드 알고리즘

- 제가 첫날에, stl에 있는 gcd쓰라고 했으면서 왜 이걸 가르치냐고 궁금하실텐데…
- 확장 유클리드 알고리즘을 배우기 위해서 입니다.
- 우리는 이제 a, b가 주어졌을 때, $as + bt = gcd(a, b)$를 만족하는 s와 t를 구할 수 있습니다.
- 이걸 어디다 쓰냐?
  - 선형 방정식 $ax + by = c$의 해를 구할 수 있다.
  - 모듈러 역원을 구할 수 있다.

확장 유클리드 알고리즘

- 아까 했던 예시 gcd(666, 396)을 가지고
  $666x + 396y = \gcd(666,396) = 18$을 만족하는 x, y를 찾아봅시다.

$$666 \div 396 = 1 \ldots 270$$

$$396 \div 270 = 1 \ldots 126$$

$$270 \div 126 = 2 \ldots 18$$

x	y	666x+396y
1	0	666
0	1	396
1	-1	666 % 396 = 270
-1	2	396 % 270 =126
3	-5	270 % 126 =18

각 값에 대한 x, y값을 저장 - 넓게 보면 dp의 방식이라고 볼 수 있음

확장 유클리드 알고리즘

- 아까 했던 예시 gcd(666, 396)을 가지고
  $666x + 396y = \gcd(666,396) = 18$을 만족하는 x, y를 찾아봅시다.

```
tuple<ll, ll, ll> xGCD(ll a, ll b){
    if (!b) return make_tuple(a, 1, 0);
    ll g, x, y;
    std::tie(g, x, y) = xGCD(b, a % b);
    return make_tuple(g, y, x - (a / b) * y);
}

int main(){
    fast_io
    auto [g, x, y] = xGCD(666, 396);
    cout << g << ' ' << x << ' ' << y << '\n';
}
```

출력 : 18 3 -5

x	y	666x+396y
1	0	666
0	1	396
1	-1	666 % 396 = 270
-1	2	396 % 270 =126
3	-5	270 % 126 =18

이항계수 : dp로 구하기

- 이항계수가 뭔지는 알고 있다고 가정하겠습니다.
- $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$ 의 성질을 이용할 것입니다. (점화식)
  - n개중에 k개를 고를 때, 1개를 안고르고 k개를 고를 수 있고, 1개를 무조건 포함시켜서 k-1개를 고를 수도 있으니 직관적인 이해가 가능합니다.

이항계수 : dp로 구하기

- bimon(n, k)를 하면 $\binom{n}{k}$ 를 구해줍니다.

- 기저 사례는
  - k == 0일 때
  - n == k일 때입니다.

```
12    ll C[1000][1000];
13
14    ll binom(int n, int k){
15        if (n < k) return 0;
16        if (k == 0 || n == k)
17            return 1;
18        ll &ret = C[n][k];
19        if (ret != -1) return ret;
20        ret = binom(n - 1, k) + binom(n - 1, k - 1);
21        return ret;
22    }
23
24    int main(){
25        fast_io
26        memset(C, -1, sizeof(C));
27    }
```

이항계수 : 모듈러 역원으로 구하기

- $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ 입니다.

  $n!, (n-k)!, k!$ 각각을 미리 구해두면 빠르게 이항계수를 구할 수 있습니다.

- 문제는 숫자가 작으면 잘 작동하지만…..
- 이항계수를 어떤 값으로 나눈 나머지을 구해보자고 한다면…. 문제가 생깁니다.

모듈로 연산 (modulo)

- 모듈러 덧셈과 뺄셈
  - $(a \mod m + b \mod m) \mod m = (a + b) \mod m$
  - Ex. $(13 \mod 10 + 16 \mod 10) \mod 10 = 29 \mod 10$
- 모듈러 곱셈
  - $(a \mod m \cdot b \mod m) \mod m = (a \cdot b) \mod m$
  - $(13 \mod 10 \cdot 16 \mod 10) \mod 10 = 208 \mod 10$
- 모듈러 연산은 무수히 많은 수를 m이라는 범위에 압축시키는 느낌입니다.

모듈로 역원 (modulo)

- 하지만 나누기 연산은 어떨까요? - **지금 우리는 정수해만을 신경씁니다.**
- $a \mod m \ / \ b \mod m = (a/b) \mod m$ 이 성립할 까요?
- Ex. $125 \mod 10 \ / \ 25 \mod 10 \neq 5 \mod 10$
  - 위 반례에서 볼 수 있듯 성립하지 않습니다..ㅠㅠ
  - 하지만 구하는 방법이 있습니다!! :)

모듈로 역원

- 0 ~ n-1까지만 있는 세상을 Z_n이라고 합시다.
- a가 주어졌을 때 $(a \cdot x) \mod n = 1$,
  이것을 만족하는 0~ n-1사이의 값 x를 구하고 싶습니다.
- 답이 없을 수도 있습니다. 예를 들면 n = 4라서 0~3까지만 있는 세상에서
  $(2 \cdot x) \mod 4 = 1$를 만족하는 x를 구해봅시다.
  - $x = 0 \rightarrow (2 \cdot x) \mod 4 = 0$
  - $x = 1 \rightarrow (2 \cdot x) \mod 4 = 2$
  - $x = 2 \rightarrow (2 \cdot x) \mod 4 = 0$
  - $x = 3 \rightarrow (2 \cdot x) \mod 4 = 2$

0~3까지 모든 수를 해봤는데 답이 없었습니다.

그 이유는 $\gcd(2,4) \neq 1$이기 때문입니다.

모듈로 역원

- $(a \cdot x) \mod n = 1$
- 위 식은 $ax + ny = 1$로 바꿀 수 있습니다.
  - 고등학교 때 배운 나머지 정리의 논리를 잘 생각해보면 왜 그런지 알 수 있죠.
  - 예를 들면 8로 나눠서 3이 되는 수는 $8 \cdot 0 + 3, 8 \cdot 1 + 3, \cdots$ 등등 이런식으로 정리되니까요
  - 따라서 a, n이 서로소 일때 확장유클리드 알고리즘을 적용하면!!
  - x와 y값을 구할 수 있고, 우리는 x값이 궁금하니 역원을 찾아낸 것입니다.

모듈로 역원

14565 - 역원(Inverse) 구하기

역원(Inverse) 구하기 성공

☆

골드 II

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	128 MB	2221	1406	1084	65.380%

문제

집합 Zn을 0부터 n-1까지의 정수 집합이라고 하자. Zn ∋ a, b, c 일 때, (a+b) mod n = 0이면 b는 a의 덧셈역이라고 하고 (a*c) mod n = 1이면 c는 a의 곱셈역이라고 한다.

정수 N, A가 주어졌을 때 Zn에서의 A의 덧셈역과 곱셈역을 구하시오.

단, 곱셈역을 구할 수 없으면 -1을 출력한다.

입력

첫 번째 줄에 N($2 \le N \le 10^{12}$)과 A($1 \le A < N$)이 주어진다.

이항계수 : 모듈러 역원으로 구하기

11401 이항 계수 3

- 그럼 다시 이항계수로 돌아와서
- $\binom{n}{k} \mod M = n! \cdot inv((n-k)!) \cdot inv(k!) \mod M$
- 아래 문제를 보면 0 ~ 1,000,000,006까지 밖에 없는 세상에서 이항계수를 구하는겁니다.

문제

자연수 N과 정수 K가 주어졌을 때 이항 계수 $\binom{N}{K}$를 1,000,000,007로 나눈 나머지를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N과 K가 주어진다. ($1 \le N \le 4,000,000$, $0 \le K \le N$)

이항계수 : 모듈러 역원으로 구하기

11401 이항 계수 3

이렇게 구조체로 저장해두면

main함수를 깔끔하게 표현할 수 있습니다.

```cpp
int main(){
    fast_io
    int n, k; cin >> n >> k;
    const int MAX = 4e6+1;
    const ll MOD = 1e9+7;
    Number_Theory<MAX, MOD> nt;
    nt.precal();
    cout << nt.binomial(n, k);
}
```

```cpp
tuple<ll, ll, ll> xGCD(ll a, ll b)
{
    if (!b)
        return make_tuple(a, 1, 0);
    ll g, x, y;
    std::tie(g, x, y) = xGCD(b, a % b);
    return make_tuple(g, y, x - (a / b) * y);
}

template <int SZ, ll B_MOD>
struct Number_Theory{
    vector<ll> fac;
    void precal(int sz = SZ){
        fac = vector<ll>(sz + 1);
        fac[0] = 1;
        for (int i = 1; i <= sz; i++){
            fac[i] = fac[i - 1] * i;
            fac[i] %= B_MOD;
        }
    }

    ll binomial(int n, int k){
        auto [g1, x1, y1] = xGCD(B_MOD, fac[n-k]);
        ll inv_nk = (y1 % B_MOD + B_MOD) % B_MOD ;
        auto [g2, x2, y2] = xGCD(B_MOD, fac[k]);
        ll inv_k = (y2 % B_MOD + B_MOD) % B_MOD;
        return fac[n] * inv_nk % B_MOD * inv_k % B_MOD;
    }
};
```

자주 쓰는 코드 Github에 저장하기

- 자주 쓰는 코드를 깃허브에 저장해두면 좋은 점이 여럿 있습니다.
  - 내 컴퓨터가 아닌 곳에서도 쉽게 볼 수 있다.
  - 나 말고 다른 사람들도 쉽게 내 코드를 참고 할 수 있다.

- 저도 오늘 배운 **이항계수, 세그먼트트리, 좌표압축 등 자주 쓰는 코드**들은 저장해두고 있습니다.

자주 쓰는 코드 Github에 저장하기

- 1. github에서 repository를 하나 만듭니다.
- 2. 로컬에서 내가 코드를 저장해둘 공간을 마련하고 repository에서 아무주소나 가지고와서 git clone 합니다.
- 3. 로컬에서 작업을 잘 한 다음에
  - git add .
  - git commit -m “내가 이번에 커밋할 내용”
  - git push
- 이렇게하면 로컬에서 작업한 다음 깃허브에 갱신됩니다.
- 궁금하면 저한테 물어봐 주세요

마무리

- 1부터 N까지의 소수를 모두 전처리하는 방법 (에라토스테네스의 체)
- 에라토스테네스의 체를 이용해서 소인수분해 하는 방법
- 그낭 유클리드 알고리즘을 이용해서 gcd를 빠르게 구하는 방법
- 확장 유클리드 알고리즘을 이용해서 선형 방정식 $ax + by = \gcd(a, b)$를 구하는 방법
- 이항계수를 dp로 구하는 방법$O(N^2)$
- 이항계수를 factorial 전처리해두고 모듈러 역원을 이용해 구하는 방법 $O(N \log N)$

마무리

추천문제

- 2960 에라토스테네스의 체
- 1850 최대공약수
- 15965 K번째 소수 : K번째 소수를 구하려면 어디까지 table을 만들어야할까?
- 1747 소수 & 팰린드롬 : 조건을 만족하는 가장 큰 소수는 무엇일까?
- 16563 어려운 소인수분해

마무리

추천문제

- 14565 역원구하기
- 13172 $\sum$: 지문이 매우 길긴한데 좋은 문제니 읽어보세요.
- 20412 추첨상 사수 대작전!(Hard)
  - 식 2개를 적절히 연산해서 a를 구할 수 있다.
  - a를 구하면 c는 걍 구한다.
- 11401 이항계수 3
- 13977 이항계수와 쿼리 : 이항계수 3을 아주 조금 응용한 문제