|  |
| --- |
| 나머지 연산.  나머지 출력하라는 문제.  문제의 정답이 너무 클 때.  씨쁠,자바 오버플로 발생.  알고리즘 문제풀이 능력.  완벽한 답을 구하고 나누라는 의미가 아니라  연산 중간중간 나머지 연산하라는 의미라고?  그래서 매번 나머지 연산을 하면 된다.  (A+\*-B)를 나머지%C 연산한 것은  A%C +\*- B%C 한 것과 같기 때문이다.  나누기의 경에는 성립하지 않는다.  뺄셈의 경우에는 음수의 경우를 고려하기 위해  +를 해줘야 한다.    대부분의 문제가 덧셈 곱셈이 많이 나오므로  두 가지의 경우에는 성립한다고 생각하면 된다.  나누기는 역원을 구해야 한다.  빼기의 나머지 연산.  프로그래밍 언어에 따라 다른 결과가 나온다.    빼기의 나머지 연산에서 더하는 이유.  나누는 수로 한 번 더해주면 된다.  나머지를 출력하라는 문제는 대부분 경우의 수를 구하는 문제.    최대 공약수  .GCD  최대공약수가 필요한 문제. 수학 문제.  말고는 없다.  중요한 알고리즘.  일단 쭉 보고, 문제 풀어보자.  나아중에 필요하면 돌아와서 정리하자.  소수 판단. 빠른 방법.  N이 소수가 되려면, 2보다 크거나 같고, N/2보다 작거나 같은 자연수로  나누어떨어지면 안된다.  소수. 가장 빠른 방법. 판별 방법. N만.  2보다 크거나 같고, 루트N보다 작거나 같은 자연수로 나누어 떨어지면 안된다.    정수로 만들기 위해.  실수는 근삿값이 최대한 안 쓰는게 좋다.  범위 안에 소수가 있는지 판별하는 방법.  루트는 수를 그다지 작게 만들지 않는다.  로그가 탁월하다.  범위 안의 모든 소수를 구하는 좋은 알고리즘.  에라토스테네스의 체.  다 쓰라고 범위내의 수를?  숫자n에 대해, n의 제곱이상의 수부터 확인하면 된다. 1부터 N이하의 n에 대해      다시 전체 bool 배열을 돌며 true/false를 확인해야 하기 때문에 사용하지 않음. |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |