암호학 HW3

2018037356 – 안동현

(아래 사진들은 코드의 캡쳐를 위해 주석을 지운 상태입니다. 본 파일에는 주석이 존재합니다)

#1 메인 함수들

1. uint64\_t mod\_add(uint64\_t a, uint64\_t b, uint64\_t m);

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

오버플로를 고려해서, a + b의 m에 대한 모듈러 연산을 해주는 함수이다.

먼저 임시 변수로 a와 b의 모듈러 연산 값을 받아두고, 해당 값으로 연산을 진행해 주었다. 이때 r1 + r2 > m 이면 m을 더해주어야 하므로 분기를 나누어야 하는데, 조건식에서 오버플로가 날 가능성이 있으므로 r1 < (m – r2)와 r1 – (m – r2) 처럼 연산의 순서를 바꾸어서 오버플로를 막아준다.

2. uint64\_t mod\_sub(uint64\_t a, uint64\_t b, uint64\_t m);

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

위 mod\_add처럼 분기를 나누어서 모듈러 연산을 리턴해주면 된다. 뺄셈에서는 오버플로가 날 가능성은 딱히 없으므로 (a – b + m)과, (a - b) 를 그대로 리턴해준다.

3. uint64\_t mod\_mul(uint64\_t a, uint64\_t b, uint64\_t m);

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이전에 과제 1에서 했던 다항식 곱셈 코드와 매우 유사하다. 하지만 조금 다르다. 해당 코드는 a에 계속 a를 더해주고(계속 곱하기 2) b를 1비트씩 왼쪽으로 밀어서, 1이라면 r에 더해주고 있는데, 이런 의미이다.

기본적으로 수를 이진수로 표현하면 1001011 이런식이다. 그리고 이는

2^6 + 0 + 0 + 2^3 + 0 + 2^1 + 2^0 과 같다. 그렇다면 a에 계속 2를 곱해주는 것이 이해가 간다. 오른쪽부터 1비트씩 돌 때마다 a\*(2^0), a\*(2^1), a\*(2^2) ……. 로 진행이 되기 때문이다.

a \* b라는 것은 결국 a를 b번 더한 것이다. (물론 2 \* 0 등이 설명이 되지 않는 등 실제 수학적 의미와는 조금 다르다.) 따라서 a에는 계속 누적으로 2를 곱해주고, b를 1비트씩 오른쪽으로 밀어서 만약 해당 비트가 1이라면 누적해 놓은 a를 r에 더해주면 되는 것이다.

위에 적어둔 이진수를 예로 들면

2^0 일 때 a\*(2^0) 를 더해주고 …….. 2^6 일 때 a\*(2^6) 을 더해주는 것이다. 이렇게 한다면 a를 b번 더해주는 것이 가능하다.

마지막에 모듈러 m을 해주지 않는 이유는 덧셈 계산중에 전부 mod m이 되었기 때문이다.

4. uint64\_t mod\_pow(uint64\_t a, uint64\_t b, uint64\_t m);

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위와 같은 논리로, a^b는 a를 b번 곱해준 것이기 때문에 mod\_add 대신 mod\_mul을 넣어주면 mod\_pow가 계산되어, a^b를 리턴해준다.

5. int miller\_rabin(uint64\_t n);

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

강의 노트에 있는 슈도 코드와는 조금 다르게 짜 보았다. miller\_rabin알고리즘은 결국 페르마의 정리를 이용하는 방법인데,

페르마 정리는 a^(p-1) ≡ 1 (mod p) (만약 p가 소수라면) 이다.

소수 판정을 할 수는 기본적으로 홀수이다. 2를 제외한 짝수는 모두 소수이기 때문이다.

따라서 n – 1 은 2^k \* q 로 나타낼 수 있다. 여기서 페르마의 정리를 이용해보면 이런 전개식을 얻을 수 있다.

(2^k \* q) = n - 1

a^(2^k \* q) == 1 mod n

a^(2^k \* q) - 1 == 0 mod n

a^(2^k \* q) - 1 = (a^(2^(k-1) \* q) + 1)(a^(2^(k-2) \* q) + 1)

......(a^q + 1)(a^q - 1)

이 곱들 중 하나라도 0이면 소수일 가능성이 있다는 것이다.

따라서 또 저 전개식이 0이 되기 위해서는 마지막 곱셈에서 a^q가 1이여야 하는 것을 제외하면 나머지는 -1 이 되어야 한다. 적어도 하나는 이를 만족해야 한다. 이때 mod n 계산이므로 n을 더해 양수로 만들면 n-1이나 1이 나와줘야 한다. 그렇다면 소수일지도 모른다는 결론이 나올 수 있다.

거기에서 이건 deterministic version 이기에 a의 값이



일 때를 전부 만족하면 무조건 소수라고 판정이 가능하다.

코드는 이러하다.

먼저 2보다 작거나, 2가 아닌 짝수는 전부 소수가 아니라고 리턴한다.

그리고 q값을 구하기 위해 q = n – 1 로 두고 홀수가 될 때까지 2로 나누어 준다.

a 배열의 길이만큼 판정을 반복하고

만약 n이 해당 배열 안의 숫자라면 바로 소수라고 판정한다.

먼저 a^q 가 1인지를 판단하고 참이면 다음 a값을 확인해준다.

이제

텍스트, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

해당 부분을 보면 tem != n – 1 && p != n – 1일 때 까지 p = mod\_mul(p, p, n);와 tem \*= 2;를 반복해주는데, 이건 tem의 경우 강의 노트 상 0 ~ k – 1 까지의 반복을 나타내기 위함이고 p의 경우

a^(2^j \* q) (단 j = 0 ~ k - 1)가 n - 1인지 확인해주는 부분이다. 2의 지수승이 계속 a의 지수로 올라가니 mod\_mul을 이용해서 자기 자신을 곱해주면 만들 수 있다.

만약 tem == n – 1이 된다면 반복문을 모두 돌았음을 의미하고

p == n – 1이 된다면 더 이상의 반복문은 의미가 없음을 의미한다.

해당 반복문이 끝났다면 tem == n - 1조건으로 끝났음을 대비해서

폰트, 텍스트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

해당 코드를 넣어주고, 아니라면 다음 a값을 확인해주는 것이다.

이제 모든 a값을 확인했을 때까지 프로그램이 끝나지 않았다면, 해당 n은 소수임을 나타내므로 PRIME을 리턴해준다.

#2 결과물

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

먼저 계산에 대한 출력물이다. 모두 문제 없이 PASSED 판정을 받는 것을 확인 할 수 있다.

패턴, 의류, 패브릭, 예술이(가) 표시된 사진

자동 생성된 설명

x = 0x8000000000000000부터 처음 100개의 소수 역시 잘 출력된다.



소수를 세는 코드 역시 잘 된다. 예상 출력에는 3957810개 이기에 뭐지? 했지만, test코드를 본 결과 3957809로 if 문을 판정하기에 3957809가 맞다고 생각했다. 시간은 162초 정도가 나와서 예상 출력보다는 다소 느리지만, 기다리지 못할 정도는 아니였다.

#3 느낀점

전반적으로 코드를 짜는데에는 어려움이 없었으나 가장 큰 문제는 miller\_rabin의 속도 문제였다..

원래 처음에는

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

해당 슈도 코드로 그대로 짰었다. k도 구하고.. mod\_pow 이용하는 등 말이다. 하지만 이렇게 하니 불필요한 반복, 계산, 조건문이 너무 많이 들어가서 그런지 속도가 상상을 초월할 정도로 느렸다. 거의 출력되는데 10분은 걸렸다.

따라서 반복, 계산, 조건을 최대한 줄이는 데에 집중한 결과가 위의 코드가 되었다. 지금은 162초대 즉 2분40초 정도에서 머문다. 그 과정에서 169 -> 167 -> 162초로 계속 줄여가는 과정이 재미가 있었다.