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);

```
uint64_t mod_add(uint64_t a, uint64_t b, uint64_t m)
{
    uint64_t r1 = a % m;
    uint64_t r2 = b % m;
    return r1 < (m - r2) ? r1 + r2 : r1 - (m - r2);
}
```

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 막아 준다.

2. uint64_t mod_sub(uint64_t a, uint64_t b, uint64_t m);

```
uint64_t mod_sub(uint64_t a, uint64_t b, uint64_t m)
{
    if (a < b) return (a-b+m) % m;
    else return (a-b) % m;
}
```

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);

```
uint64_t mod_mul(uint64_t a, uint64_t b, uint64_t m)
{
    uint64_t r = 0;
    while (b > 0){
        if (b & 1) r = mod_add(r, a, m);
        b = b >> 1;
        a = mod_add(a,a,m);
    }
    return r;
}
```

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);

```
uint64_t mod_pow(uint64_t a, uint64_t b, uint64_t m)
{
    uint64_t r = 1;
    while (b > 0){
        if (b & 1) r = mod_mul(r,a,m);
        b = b >> 1;
        a = mod_mul(a,a,m);
    }
    return r;
}
```

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);

```
int miller_rabin(uint64_t n)
{
    uint64_t q = n - 1, tem, p, a_val;

    if (n < 2 || (n != 2 && n % 2 == 0)) return COMPOSITE;

    while (q % 2 == 0) q /= 2;

    for (int i = 0; i < BASELEN; i++){
        if((a_val = a[i]) == n) return PRIME;

        tem = q;

        if ((p = mod_pow(a_val, tem, n)) == 1) continue;

        while(tem != n - 1 && p != n - 1){
            p = mod_mul(p, p, n);
            tem *= 2;
        }
        if (p != n - 1){
            return COMPOSITE;
        }
    }

    return PRIME;
}
```

TEST (n)

Find integers k, q, $k > 0$, q odd, so that $(n-1) = 2^k q$

Select a random integer a, $1 < a < n-1$

if $a^q \bmod n = 1$ **then** return ("inconclusive");

for $j = 0$ to $k - 1$ do

  if $((a^q)^{2^j} \bmod n = n\text{-}1)$ **then**

    return ("inconclusive")

return ("composite")

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

```
const uint64_t a[BASELEN] = {2,3,5,7,11,13,17,19,23,29,31,37};
```

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

```
tem = q;

if ((p = mod_pow(a_val, tem, n)) == 1) continue;

while(tem != n - 1 && p != n - 1){
    p = mod_mul(p, p, n);
    tem *= 2;
}
```

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

```
if (p != n - 1){
    return COMPOSITE;
}
```

해당 코드를 넣어주고, 아니라면 다음 a값을 확인해주는 것이다.

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

#2 결과물

```
wollong@wollong-virtual-machine:~/proj#3$ ./test
a = 13053660249015046863, b = 14731404471217122002, m = 16520077267041420904
a+b mod m = 11264987453190747961.....PASSED
a-b mod m = 14842333044839345765.....PASSED
a*b mod m = 13008084103192797750.....PASSED
a^b mod m = 12523224429397597497.....PASSED
a = 18446744073709551615, b = 72057594037927935, m = 65536
a+b mod m = 65534.....PASSED
a-b mod m = 0.....PASSED
a*b mod m = 1.....PASSED
a^b mod m = 65535.....PASSED
18446744073709551613^18446744073709551613 mod 5 = 3.....PASSED
2 3 5 7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97 101 103 107 109 113
127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
233 239 241 251 257 263 269 271 277 281
283 293 307 311 313 317 331 337 347 349
353 359 367 373 379 383 389 397 401 409
419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509 521 523 541
```

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

```
9223372036854775837 9223372036854775907 9223372036854775931 9223372036854775939
9223372036854775963 9223372036854776063 9223372036854776077 9223372036854776167
9223372036854776243 9223372036854776257 9223372036854776261 9223372036854776293
9223372036854776299 9223372036854776351 9223372036854776393 9223372036854776407
9223372036854776561 9223372036854776657 9223372036854776687 9223372036854776693
9223372036854776711 9223372036854776803 9223372036854777017 9223372036854777059
9223372036854777119 9223372036854777181 9223372036854777211 9223372036854777293
9223372036854777341 9223372036854777343 9223372036854777353 9223372036854777359
9223372036854777383 9223372036854777409 9223372036854777433 9223372036854777463
9223372036854777509 9223372036854777517 9223372036854777653 9223372036854777667
9223372036854777721 9223372036854777803 9223372036854777853 9223372036854778027
9223372036854778037 9223372036854778129 9223372036854778171 9223372036854778193
9223372036854778291 9223372036854778307 9223372036854778331 9223372036854778351
9223372036854778421 9223372036854778447 9223372036854778487 9223372036854778637
9223372036854778739 9223372036854778897 9223372036854778973 9223372036854778997
9223372036854779053 9223372036854779081 9223372036854779099 9223372036854779149
9223372036854779173 9223372036854779339 9223372036854779351 9223372036854779357
9223372036854779459 9223372036854779491 9223372036854779591 9223372036854779627
9223372036854779633 9223372036854779663 9223372036854779731 9223372036854779753
9223372036854779789 9223372036854779813 9223372036854779831 9223372036854779891
9223372036854779953 9223372036854779971 9223372036854780017 9223372036854780031
9223372036854780073 9223372036854780089 9223372036854780097 9223372036854780139
9223372036854780163 9223372036854780169 9223372036854780193 9223372036854780199
9223372036854780239 9223372036854780241 9223372036854780251 9223372036854780283
9223372036854780397 9223372036854780551 9223372036854780611 9223372036854780647
```

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

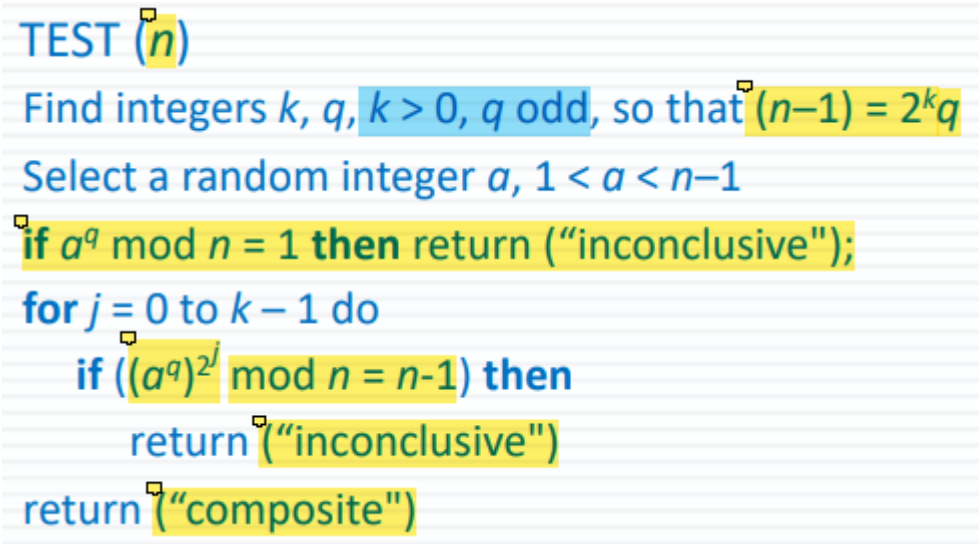
```
x = 1부터 67108864까지 소수를 세는 중.....................................................
.....소수 개수: 3957809개.....PASSED
계산 시간: 162.6988초
```

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



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.