Project#1

2018044893 이수정

<함수설명>

int gcd(int a, int b)

```c
int gcd(int a, int b)
{
        int tmp1;
        while(b != 0)//  b == 0일 때 나누어 떨어진 경우이기 때문에 a가 최대공약수가 됨.
        {
                tmp1 =a%b;
                a = b;
                b = tmp1;
        }
        return a;
    // 여기를 완성하세요
}
```

유클리드 알고리즘 gcd(a,b) = gcd(b,a%b)이며 b == 0일 때 나누어 떨어진 경우이기 때문에 최대공약수 a가 return되도록 구현하였다.

int xgcd(int a, int b, int *x, int *y)

```c
int xgcd(int a, int b, int *x, int *y) //x와 y값을 가져오기 위해 포인터를 잘 확인 해야함.
{
        int x0,x1,y0,y1,d0,d1;
        int q,tmp2;
        x0=y1=1;// x0 = 1, y1 = 1
        x1=y0=0;// x1 = 0, y0 = 0
        d0 = a;
        d1 = b;
        while(d1!=0)
        {
                //gcd(a,b) = a * x1 + b * y1을 만족하는 x1과 y1값을 구함
                q = d0/d1;

                tmp2 = x0 - q*x1;//x[i+1] = x[i-1] - q[i]*x[i]
                x0 = x1;
                x1 = tmp2;

                tmp2 = y0 - q*y1;//y[i+1] = y[i-1] - q[i]*y[i]
                y0 = y1;
                y1 = tmp2;

                tmp2 = d0 - q*d1;//d[i+1] = d[i-1] - q[i]*d[i]
                d0 = d1;
                d1 = tmp2;
        }
        *x = x0;
        *y = y0;
        return d0;
    // 여기를 완성하세요
}
```

유클리드 알고리즘 gcd(a,b) = gcd(b,a%b) = gcd(a%b,b%(a%b)) 에서 a =

d0,b = d1, a%b = d2, b%(a%b) = d3라고 할 때,

$d2 = d0 - (d0/d1)*d1$,

$d3 = d1 - (d1/d2)*d2$라 할수있다.

이를 이용하여 $d(n+1) = d(n-1) - (d(n-1)/d(n)) * d(n)$

$x(n+1) = x(n-1) - (d(n-1)/d(n)) * x(n)$

$y(n+1) = y(n-1) - (d(n-1)/d(n)) * y(n)$를 통해

$ax + by = gcd(a, b)$을 만족하는 x,y를 구할수 있다.

int mul_inv(int a, int m)

```
int mul_inv(int a, int m)
{
        int d0,d1,x0,x1,q,tmp3;
        d0 = a;
        d1 = m;
        x0 = 1;
        x1 = 0;
        while(d1>1)
        {
                q = d0/d1;

                tmp3 = d0 - q*d1;//변형된 확장유클리드 알고리즘과 swap(d0,d1)
                d0 = d1;
                d1 = tmp3;

                tmp3 =x0 - q*x1;
                x0 = x1;
                x1 = tmp3;
        }
        if(d1 == 1)     return(x1>0 ? x1 : x1+m);
        else return 0;


    // 여기를 완성하세요
}
```

$gcd(a, b) = ax + by$ 에서 만약 a,b가 서로소라면 $gcd(a,b) = dn = 1$이다

즉, $1 = a*x(n) + b* y(n)$이며 모든 항에 mod b를 해준다면

$1 = ax(n) \bmod b + 0 \Leftrightarrow x(n) = a^{(-1)} \bmod b$가 된다.

모든 항에 mod a를 해주면 $x(n) = b^{(-1)} \bmod a$임을 알 수있다.

$d0 = a$, $d1 = m$, $x0 = 1$, $x1 = 0$이라 정의하고 확장 유클리드 알고리즘을 이용하여 역원을 구하였다.

uint64_t umul_inv(uint64_t a, uint64_t m)

확장 유클리드 알고리즘 int gcd(int a, int b)
int xgcd(int a, int b, int x, int y) 를

통해 $\underbrace{gcd(a,b)}_{d_0} = \underbrace{gcd(b, a\%b)}_{d_1} = \underbrace{gcd(a\%b, b\%(a\%b))}_{d_2, d_3}$

$$\begin{pmatrix} d_2 = d_0 - (d_0/d_1)\cdot d_1 \\ d_3 = d_1 - (d_1/d_2)\cdot d_2 \\ \vdots \\ d_{k+1} = d_{k-1} - (d_{k-1}/d_k)\cdot d_k \end{pmatrix}$$

$d_{k+1} = 0$ 일때 $gcd(a,b) = d_k$ ④

$d_{k+1} = d_{k-1} - (d_{k-1}/d_k)\cdot d_k$ ①

$gcd(a,b) = d_k = a x_k + b y_k$ 임을 알고있다. ②

$\langle x_k = a^{-1} \bmod m$ 유도$\rangle$

$b=m$ 이라고 하고 ②에 대입해본다면,

$d_k = a x_k + m y_k$ ... ③

$d_k \bmod m = a x_k \bmod m + m y_k \bmod m$ ... ③$\cdot \bmod m$

a와 b가 서로소라면 $d_k = 1$, ... ⑤

$1 = a x_k \bmod m + 0 \Leftrightarrow a^{-1} \bmod m = x_k$

즉 d_k 가 최대공약수가 될때, x_k 값은 $a^{-1} \bmod m$ 이 된다

x_k가 음수라면 {예) $-1 \bmod 11 = (11-1) \bmod 11$} 모듈로 성질을 이용하여 m을 더해준다 ... ⑥

```c
uint64_t umul_inv(uint64_t a, uint64_t m) // unsigned 64 비트 정수에서 mul_inv와 같은 방식을 사용>
하여 a^-1 mod m을 구함
{
        uint64_t d0,d1,x0,x1,q,tmp4;
        d0 = a;
        d1 = m;
        x0 = 1;
        x1 = 0;
        while(d1>1)
        {
                q = d0/d1;

                tmp4 = d0 - q*d1;
                d0 = d1;
                d1 = tmp4;

                tmp4 = x0 - q*x1;
                x0 = x1;
                x1 = tmp4;
        }
        if(d1 == 1)     return((x1>>63)==0 ? x1 : x1+m);//첫번째 비트를 통해 부호를 확인함.
        else    return 0;

    // 여기를 완성하세요
}
```

→ ④

① 이용

→ ⑤

⑥

→ 역이 존재하지 X

mul_inv함수와 동일한 방법을 사용하였고 unsigned int 64비트는 첫번째 비트가 양수와 음수를 결정하기 때문에 63비트 앞으로 이동하여 구분하였다.

uint8_t gf8_mul(uint8_t a, uint8_t b)

```c
uint8_t gf8_mul(uint8_t a, uint8_t b)
{
        uint8_t res = 0;
        while(b > 0)
        {
                if(b & 1)       res = res ^ a; // b가 1일때 a값이 존재하므로 res와 XOR해준다.
                b = b >> 1; // b를 한비트씩 줄여줌
                a = ((a<<1)^((a>>7) & 1 ? 0X1B : 0));// a가 7차식인경우 atimes를 통해 ax로 만들어준다.
                //x^8 = x^4+x^3+x+1 (mod x^8+x^4+x^3+x+1) 특성을 이용하여 x^4+x^3+x+1 = 0X1B를 XOR해준다.
        }
        return res;
    // 여기를 완성하세요
}
```

$m(x) = x^8 + x^4 + x^3 + x + 1$

$x*8 \bmod m(x) = x^4 + x^3 + x + 1$ 이므로 a가 7차식인가를 확인하여 7차식인경우 atimes를 통해 ax로 만들어주고 $x^4 + x^3 + x + 1$인 11011(B)를 XOR 한다.

그 때 해당하는 비트b가 1인경우의 m(x)로 모듈러 된 모든항을 더해주는 방법이다.

uint8_t gf8_pow(uint8_t a, uint8_t b)

```c
uint8_t gf8_pow(uint8_t a, uint8_t b) // project#1.pdf를 참고하여 일반적인 square multiplication을 이해하였다.
{
        uint8_t res = 1;
        while(b>0)
        {
                if(b & 1) res = gf8_mul(res,a); //예) b = x^7+x^4+x^3+x+1일 때,
                                                //b = 10011011(B),      result = result * a^(2^0),
                                                //b = 1001101(B),       result = result * a^(2^1),
                                                //b = 10011(B),         result = result * a^(2^3),
                                                //b = 1001(B),  result = result * a^(2^4),
                                                //b = 1(B),             result = result * a^(2^7),
                                                //b = 2^n,      result = result * a^(2^n)

                a = gf8_mul(a,a);
                //a^(2^n) * a^(2^n) = a^(2^(n+1)) , gf8_mul을 이용하여 a*a mod x^8+x^4+x^3+x+1를 구함
                b = b >> 1;
        }
        return res;
    // 여기를 완성하세요
}
```

// project#1.pdf를 참고하여 일반적인 square multiplication을 이해하였고 이를 gf8_mul과 혼용하였다.

예) $b = x^7 + x^4 + x^3 + x + 1$이라 할 때 b = 10011011(B)

b = 10011011(B) = $2^7 + 2^4 + 2^3 + 2 + 1$ 이므로

$a^b = a^{(2^7)} * a^{(2^4)} * a^{(2^3)} * a^{(2^1)} * a^{(2^0)}$으로 된다.

따라서 $a^b \bmod m(x) = a^{(2^7)} \bmod m(x) * a^{(2^4)} \bmod m(x) * a^{(2^3)}$

mod m(x) * a^(2^1) mod m(x) * a^(2^0) mod m(x)을 이용하였다.

a^(2^n) = a^(2^(n-1)) * a(2^(n-1))이므로 b를 한비트 씩 줄일때마다 a*a 의 모듈로 값인 gf8_mul(a,a)를 a로 갱신해주는 square multiplication의 구조를 활용하였다.

<컴파일 과정>

환경 : ubuntu-20.04.3-desktop-amd64(리눅스)

```
soo@soo-VirtualBox:~/Documents$ gcc -o euclid_gf8 euclid_gf8.c -lbsd
euclid_gf8.c: In function ‘main’:
euclid_gf8.c:301:28: warning: format ‘%llu’ expects argument of type ‘long long unsigned int’, b
ut argument 3 has type ‘uint64_t’ {aka ‘long unsigned int’} [-Wformat=]
  301 |     printf("a = %d, m = %llu, a^-1 mod m = %llu", a, m, ai);
      |                         ~~~^                          ~
      |                            |                          |
      |                            long long unsigned int     uint64_t {aka long unsigned int}
      |                         %lu
euclid_gf8.c:301:47: warning: format ‘%llu’ expects argument of type ‘long long unsigned int’, b
ut argument 4 has type ‘uint64_t’ {aka ‘long unsigned int’} [-Wformat=]
  301 |     printf("a = %d, m = %llu, a^-1 mod m = %llu", a, m, ai);
      |                                            ~~~^         ~~
      |                                               |         |
      |                                               |         uint64_t {aka long unsigned int}
      |                                               long long unsigned int
      |                                            %lu
euclid_gf8.c:310:28: warning: format ‘%llu’ expects argument of type ‘long long unsigned int’, b
ut argument 3 has type ‘uint64_t’ {aka ‘long unsigned int’} [-Wformat=]
  310 |     printf("a = %d, m = %llu, a^-1 mod m = %llu", a, m, ai);
      |                         ~~~^                          ~
      |                            |                          |
      |                            long long unsigned int     uint64_t {aka long unsigned int}
      |                         %lu
euclid_gf8.c:310:47: warning: format ‘%llu’ expects argument of type ‘long long unsigned int’, b
ut argument 4 has type ‘uint64_t’ {aka ‘long unsigned int’} [-Wformat=]
  310 |     printf("a = %d, m = %llu, a^-1 mod m = %llu", a, m, ai);
      |                                            ~~~^         ~~
      |                                               |         |
      |                                               |         uint64_t {aka long unsigned int}
      |                                               long long unsigned int
      |                                            %lu
euclid_gf8.c:319:28: warning: format ‘%llu’ expects argument of type ‘long long unsigned int’, b
ut argument 3 has type ‘uint64_t’ {aka ‘long unsigned int’} [-Wformat=]
  319 |     printf("a = %d, m = %llu, a^-1 mod m = %llu", a, m, ai);
      |                         ~~~^                          ~
      |                            |                          |
      |                            long long unsigned int     uint64_t {aka long unsigned int}
      |                         %lu
euclid_gf8.c:319:47: warning: format ‘%llu’ expects argument of type ‘long long unsigned int’, b
ut argument 4 has type ‘uint64_t’ {aka ‘long unsigned int’} [-Wformat=]
  319 |     printf("a = %d, m = %llu, a^-1 mod m = %llu", a, m, ai);
      |                                            ~~~^         ~~
      |                                               |         |
      |                                               |         uint64_t {aka long unsigned int}
      |                                               long long unsigned int
      |                                            %lu
soo@soo-VirtualBox:~/Documents$
```

리눅스 환경에서 %llu를 사용할 때 정의 된 변수가 64비트를 모두 사용하지 않고 32비트로 커버가 가능한 경우 %lu를 이용하도록 warning이 생긴다.

이 경우 %lu로 바꾸어 주거나 64비트 사용 매크로를 정의하여 %llu대신 %“PRIu64”를 이용하면 warning문제를 해결할 수 있다.

-define macros

```
#define __STDC_FORMAT_MACROS
#include <inttypes.h>
```

-해당부분 코드

```
    printf("--- 기본 umul_inv 시험 ---\n");
    a = 5; m = 9223372036854775808u;
    ai = umul_inv(a, m);
    printf("a = %d, m = %"PRIu64", a^-1 mod m = %"PRIu64"", a, m, ai);
    if (ai != 5534023222112865485u) {
        printf(" <- inversion error\n");
        exit(1);
    }
    else
        printf(" OK\n");
    a = 17; m = 9223372036854775808u;
    ai = umul_inv(a, m);
    printf("a = %d, m = %"PRIu64", a^-1 mod m = %"PRIu64"", a, m, ai);
    if (ai != 8138269444283625713u) {
        printf(" <- inversion error\n");
        exit(1);
    }
    else
        printf(" OK\n");
    a = 85; m = 9223372036854775808u;
    ai = umul_inv(a, m);
    printf("a = %d, m = %"PRIu64", a^-1 mod m = %"PRIu64"", a, m, ai);
    if (ai != 9006351518340545789u) {
        printf(" <- inversion error\n");
                                                        320,39        97%
```

수정 이후 컴파일과정

```
soo@soo-VirtualBox:~/Documents$ make
gcc -c euclid_gf8.c
gcc -o euclid_gf8 euclid_gf8.o -lbsd
soo@soo-VirtualBox:~/Documents$
```

<실행 결과물>

```
soo@soo-VirtualBox:~/Documents$ ./euclid_gf8
--- 기본 gcd 시험 ---
gcd(28,0) = 28
gcd(0,32) = 32
gcd(41370,22386) = 42
gcd(22386,41371) = 1
--- 기본 xgcd, mul_inv 시험 ---
42 = 41370 * -204 + 22386 * 377
41370^-1 mod 22386 = 0, 22386^-1 mod 41370 = 0
1 = 41371 * 4285 + 22386 * -7919
41371^-1 mod 22386 = 4285, 22386^-1 mod 41371 = 33452
--- 무작위 mul_inv 시험 ---
................No error found
--- GF(2^8)에서 기본 a*b 시험 ---
28 * 7 = 84
127 * 68 = 21
--- GF(2^8)에서 전체 a*b 시험 ---
..................................................................................................
..................................................................................................
...................................................................No error found
--- 기본 umul_inv 시험 ---
a = 5, m = 9223372036854775808, a^-1 mod m = 5534023222112865485 OK
a = 17, m = 9223372036854775808, a^-1 mod m = 8138269444283625713 OK
a = 85, m = 9223372036854775808, a^-1 mod m = 9006351518340545789 OK
Congratulations!
soo@soo-VirtualBox:~/Documents$
```

[공지]에 올라온unsigned 64 비트 곱의 역 샘플 출력 결과

```
for test umul_inv
a = 1, m = 0, a^-1 mod m = 1
a = 2, m = 0, a^-1 mod m = none
a = 3, m = 0, a^-1 mod m = 3074457345618258603
a = 4, m = 0, a^-1 mod m = none
a = 5, m = 0, a^-1 mod m = 5534023222112865485
a = 6, m = 0, a^-1 mod m = none
a = 7, m = 0, a^-1 mod m = 7905747460161236407
a = 8, m = 0, a^-1 mod m = none
a = 9, m = 0, a^-1 mod m = 1024819115206086201
a = 10, m = 0, a^-1 mod m = none
a = 11, m = 0, a^-1 mod m = 3353953467947191203
a = 12, m = 0, a^-1 mod m = none
a = 13, m = 0, a^-1 mod m = 5675921253449092805
a = 14, m = 0, a^-1 mod m = none
a = 15, m = 0, a^-1 mod m = 7993589098607472367
a = 16, m = 0, a^-1 mod m = none
a = 17, m = 0, a^-1 mod m = 8138269444283625713
a = 18, m = 0, a^-1 mod m = none
a = 19, m = 0, a^-1 mod m = 485440633518672411
a = 20, m = 0, a^-1 mod m = none
a = 21, m = 0, a^-1 mod m = 5709706499005337405
a = 22, m = 0, a^-1 mod m = none
a = 23, m = 0, a^-1 mod m = 6015242632731375527
a = 24, m = 0, a^-1 mod m = none
a = 25, m = 0, a^-1 mod m = 1106804644422573097
a = 26, m = 0, a^-1 mod m = none
a = 27, m = 0, a^-1 mod m = 341606371735362067
a = 28, m = 0, a^-1 mod m = none
a = 29, m = 0, a^-1 mod m = 3816567739388183093
a = 30, m = 0, a^-1 mod m = none
a = 31, m = 0, a^-1 mod m = 8033259515970288607
a = 32, m = 0, a^-1 mod m = none
a = 33, m = 0, a^-1 mod m = 1117984489315730401
a = 34, m = 0, a^-1 mod m = none
a = 35, m = 0, a^-1 mod m = 3425823899403202443
a = 36, m = 0, a^-1 mod m = none
a = 37, m = 0, a^-1 mod m = 1495681951922396077
a = 38, m = 0, a^-1 mod m = none
a = 39, m = 0, a^-1 mod m = 8040888442386214807
a = 40, m = 0, a^-1 mod m = none
a = 41, m = 0, a^-1 mod m = 1124801467909119001
a = 42, m = 0, a^-1 mod m = none
a = 43, m = 0, a^-1 mod m = 214497024112901763
a = 44, m = 0, a^-1 mod m = none
a = 45, m = 0, a^-1 mod m = 5738987045154082725
a = 46, m = 0, a^-1 mod m = none
```

a = 255까지 모두 확인함.

[공지]에 올라온 umul_inv() 함수 검증 법 결과

```
soo@soo-VirtualBox:~/Documents$ ./euclid_gf8
--- 기본 gcd 시험 ---
gcd(28,0) = 28
gcd(0,32) = 32
gcd(41370,22386) = 42
gcd(22386,41371) = 1
--- 기본 xgcd, mul_inv 시험 ---
42 = 41370 * -204 + 22386 * 377
41370^-1 mod 22386 = 0, 22386^-1 mod 41370 = 0
1 = 41371 * 4285 + 22386 * -7919
41371^-1 mod 22386 = 4285, 22386^-1 mod 41371 = 33452
--- 무작위 mul_inv 시험 ---
................No error found
--- GF(2^8)에서 기본 a*b 시험 ---
28 * 7 = 84
127 * 68 = 21
--- GF(2^8)에서 전체 a*b 시험 ---
................................................................................
................................................................................
................................................................................
...............No error found
--- 기본 umul_inv 시험 ---
a = 5, m = 9223372036854775808, a^-1 mod m = 5534023222112865485 OK
a = 17, m = 9223372036854775808, a^-1 mod m = 8138269444283625713 OK
a = 85, m = 9223372036854775808, a^-1 mod m = 9006351518340545789 OK
for test umul_inv
not error
Congratulations!
soo@soo-VirtualBox:~/Documents$
```

'Not error'에 해당하는 부분 test_umul_inv-2.c을 활용한 함수에 대한 결과값이다.

<소감/어려웠던점>

```
int xgcd(int a, int b, int *x, int *y)
{
        int x1,x2,y1,y2;
        int tmp_x,tmp_y,tmp2,q;
        x1=y2=1;
        x2=y1=0;
        if(a<b)
        {
                tmp2 = a;
                a = b;
                b =tmp2;
        }
        d1 = a;
        d2 = b;
        while(b>0)
        {
                q = a/b;
                tmp_x = x1 - q*x2;
                x1 = x2;
                x2 = tmp_x;

                tmp_y = y1 - q*y2;
                y1 = y2;
                y2 = tmp_y;

                tmp2 = d1 - q*d2;
                d1 = d2;
                d2 = tmp2;
        }
        *x = x1;
        *y = y1;
        return a;
    // 여기를 완성하세요
}
```

```
--- 기본 gcd 시험 ---
gcd(28,0) = 28
gcd(0,32) = 32
gcd(41370,22386) = 42
gcd(22386,41371) = 1
--- 기본 xgcd, mul_inv 시험 ---
42 = 41370 * -204 + 22386 * 377
41370^-1 mod 22386 = 0, 22386^-1 mod 41370 = 0
1 = 41371 * 4285 + 22386 * -7919
41371^-1 mod 22386 = 4285, 22386^-1 mod 41371 = 33452
--- 무작위 mul_inv 시험 ---
Inversion error
soo@soo-VirtualBox:~/Documents$
```

초반 xgcd함수에서 return 값을 잘못 입력하여 무작위 mul_inv시험에서 exit()되었다. 이때 출력내용과 예상출력만 비교하고 넘어간 것이 문제였고 이미 완성한 코드들도 한번씩 다시 체크할 계기가 되어 굳이 사용 할 필요 없는 수식은 삭제할 수 있었다.

xgcd함수의 코드를 수정하는 과정에서 이러한 알고리즘이 나오게 된 과정을 직접 따라가 보았고 이를 활용하는 mul_inv, umul_inv 함수를 완성하는 과정에 많은 도움이 되었다.

```
        if(d1 == 1)     return(x1>0 ? x1 : x1+m);
        else return 0;


    // 여기를 완성하세요
}
```

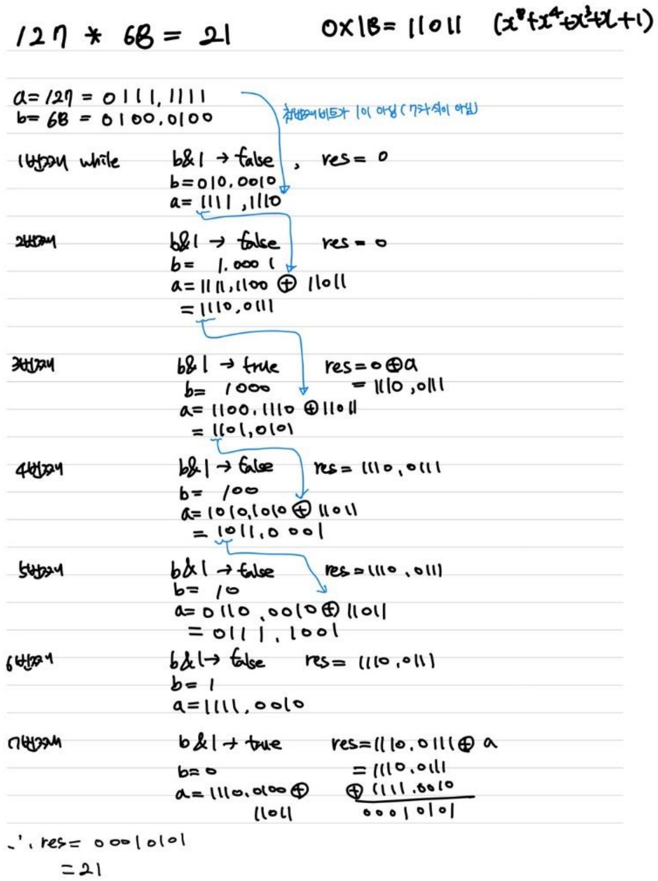
```
        if(d1 == 1)     return((x1>>63)==0 ? x1 : x1+m);//첫번째 비트를 통해 부호를 확인함.
        else    return 0;

    // 여기를 완성하세요
}
```

umul_inv같은경우 mul_inv와 같은 알고리즘을 사용하였지만 비교연산자를 통해 양수 음수를 확인할 수 없었다는 점에서 어려움이 있었다. 처음에 단순히 int64_t result = x1를 통해 양수,음수를 비교하여 하였지만 gf8_mul과 gf8_pow함수를 이해하면서 비트연산에 이해가 생기면서 첫번째 비트를 확인하는 방법으로 바꾸었다.

64비트를 사용하는 역원을 추가로 구현하는 이유는 대부분의 암호학 알고리즘은 64비트를 따르기 때문에 사용되었다고 생각한다. 확장 유클리드 알고리즘을 이용한 암호화라는 말이 글로는 와닿지 않았지만 직접 코딩하는 과정을 통해 이해되었다.

```
rmat=]
  319 |     printf("a = %d, m = %llu, a^-1 mod m = %llu", a, m, ai);
      |                                            ~~~^          ~~
      |                                               |          |
      |                                               |          uint64_t {aka long un
signed int}
      |                                             long long unsigned int
      |                                            %lu
```

리눅스 환경에서 과제를 진행하였다. 이때 uint64_t를 long unsigned int로 인식하여 프린트 함수에서 %llu에 대한 warning이 뜨는 문제가 발생하였다. 64비트 매크로를 정의하여 %llu대신 %“PRIu64”를 이용하면 warning 문제를 해결하였다.



127 * 6B = 21   0x1B = 11011 ($x^8+x^4+x^3+x+1$)

a = 127 = 0111,1111
b = 6B = 0100,0100
최상위비트가 1이 아님 (7차식이 아님)

1번째 while b&1 → false , res = 0
b = 010,0010
a = 1111,1110

2번째 b&1 → false res = 0
b = 1,0001
a = 1111,1100 ⊕ 11011
= 1110,0111

3번째 b&1 → true res = 0⊕a
b = 1000 = 1110,0111
a = 1100,1110 ⊕11011
= 1101,0101

4번째 b&1 → false res = 1110,0111
b = 100
a = 1010,1010 ⊕ 11011
= 1011,0001

5번째 b&1 → false res = 1110,0111
b = 10
a = 0110,0010 ⊕ 11011
= 0111,1001

6번째 b&1 → false res = 1110,0111
b = 1
a = 1111,0010

7번째 b&1 → true res = 1110,0111 ⊕ a
b = 0 = 1110,0111
a = 1110,0100 ⊕ 11011 ⊕ 1111,0010
0001,0101

∴ res = 0001,0101
= 21

gf8_mul()함수가 가장 이해하기 어려웠기 때문에 강의노트에 나온 예시인 127 * 68 = 21을 통해 계산하는 과정을 통해 이해할 수 있었다.

```
r = 1;
while (b > 0) {
    if (b & 1)
        r = r * a;
    b = b >> 1;
    a = a * a;
}
return r;
```

gf8_pow() 함수는 강의에서 배운 모듈로의 특성을 생각하여 각 항을 모듈러 계산한 후 연산하는 방법을 이용하였고 제시된 gf_mul() 함수와 square multiplication을 통해 쉽게 접근이 가능하였다.