REPORT

암호학 프로젝트3 Miller-Rabin

과목명	암호학
담당교수	오희국 교수님
학생이름	지현도
학과	컴퓨터학부
학번	2021004866
제출일	2023/10/11

HANYANG UNIVERSITY

*본인이작성한함수에대한설명:

1. uint64_t mod_add(uint64_t a, uint64_t b, uint64_t m)

```
15 uint64_t mod_add(uint64_t a, uint64_t b, uint64_t m)
16 {
17     a = a % m;
18     b = b % m;
19     if(a>=m-b)
20         return a-(m-b); // if(a+b>=m) -> if(a>=m-b)
21     return a+b;
22 }
```

모듈러 연산에서는 분배법칙이 성립되므로 미리 a와b에 m으로 모듈러연산을 적용해주어 m보다 작은 수로 만들어 준 후 계산을 진행하였다.

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

```
28 uint64_t mod_sub(uint64_t a, uint64_t b, uint64_t m)
29 {
30     a = a % m;
31     b = b % m;
32     if(a < b)
33         return m-(b-a); // == return a-b+m -> m-(b-a)
34     return a-b;
35 }
```

Mod_add와 마찬가지로 미리 모듈러연산을 적용해 준 후 계산해주었다.

3. uint64_t mod_mul(uint64_t a, uint64_t b, uint64_t m)

```
49 uint64_t mod_mul(uint64_t a, uint64_t b, uint64_t m)
50 {
51     uint64_t r = 0;
52         while(b > 0){
53             if(b & 1)
54                 r = mod_add(r, a, m);
55             b = b >> 1;
56             a = mod_add(a, a, m);
57         }
58         return r;
59 }
```

교수님이 미리 제공해주신 함수를 사용하였습니다.

4. uint64_t mod_pow(uint64_t a, uint64_t b, uint64_t m)

```
73 uint64_t mod_pow(uint64_t a, uint64_t b, uint64_t m)
74 {
75     uint64_t r = 1;
76         while(b > 0){
77             if(b & 1)
78                 r = mod_mul(r, a, m);
79             b = b >> 1;
80             a = mod_mul(a, a, m);
81         }
82         return r;
83 }
```

위 함수 또한 교수님이 제공해주신 함수를 사용하였습니다.

이때 mod_mul과 원리는 같지만 a^b는 곧 a를 b번 곱한것이므로 r=mod_add가 아닌 r=mod_mul를 사용하였다.

5. int isComposite(uint64_t a, uint64_t b, uint64_t k, uint64_t m)

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")

위와 같이 교수님의 강의노트를보면 페르마의 정리를 활용하여 소수의 확률을 계산하는 pseudocode가 제시되어있다.

이를 참고했을때

q, k는 n이 홀수일때(n-1)=2^k*q를 만족한다.

이때, j가 0<=j<k일때 (a^q)^2^j=1(mod n)이라면 소수일 확률이 크고(inconclusive), 아니라면 n은 합성수(composite)라고 볼 수 있다.

```
/*
 * isComposite() - Test(n), n이 소수인지 합성수인지 확인(소수일확률이 높다 - 꼭 소수라는 것은 아니다)
 * 만약 합성수라면 1을 반환하고, 아니라면 0을 반환한다
 *
 */
int isComposite(uint64_t a, uint64_t q, uint64_t k, uint64_t n){
    uint64_t tmp = mod_pow(a, q, n);
    if(tmp == 1 || tmp == n-1)
        return 0;
    for(uint64_t j=1; j<k; j++){
        tmp = mod_mul(tmp,tmp,n);
        if(tmp == n-1)
            return 0;
    }
    return 1;
}
```

6. int millder_rabin(uint64_t n)

```
120 int miller_rabin(uint64_t n)
121 {
122     if(n == 2)
123         return 1;
124         else if(n < 2 || n % 2 == 0)
125             return 0;
126         uint64_t k = 0, q = n-1;
127         while(!(q & 1)){ // (n-1) = (2^k)q 인 k와 q를 찾을때까지 비트를 밀어가며 반복 계산
128             k++;
129             q = q >> 1;
130         }
131         for(int i=0; (i<BASELEN && a[i]<n-1); i++){
132             if(isComposite(a[i], q, k, n))
133                 return 0;
134         }
135         return 1;
136 }
```

n이 소수라면(소수일 확률이 높다면) 1을 반환하고, 합성수라면 0을 반환한다.

이때 n=2라면 소수이고, 2보다 크면서 짝수여도 합성수이다.

위 조건을 피해가는 즉 n이 2보다 크면서 홀수일 경우에는 밀러라빈 알고리즘을 적용하여 계산해준다.

(n-1)=2^k*q 를 만족하는 정수 k,q를 찾아야 한다.

우선 q를 n-1로 초기값을 잡고, 2로 나누어지지 않을때까지 나눈다. 나눌때마다 횟수를 증가시켜 총 횟수가 k, 남은 수는 q가 된다고 할 수 있다. 이때 n이 2^64보다 작다는 가정이 있으므로 a[]배열의 숫자중에서 n이 합성수라는 계산이 나오지 않는다면 n은 소수라고 판단할 수 있다.(소수일 확률이 높

다)

그 후 위에서 구현한 isComposite()를 이용하여 1<a<n-1를 만족하는 구간에서 합성수가 있는지 확인하고 만약 모든 a가 n이 소수일 확률이 크다고(합성수가아니다라고) 계산되면 1을 반환한다.

*컴파일 과정

```
[jihyeondo@192 proj#3 % make
gcc -Wall -O3 -Xpreprocessor -fopenmp -c miller_rabin.c
gcc -o test test.o miller_rabin.o  -lomp
```

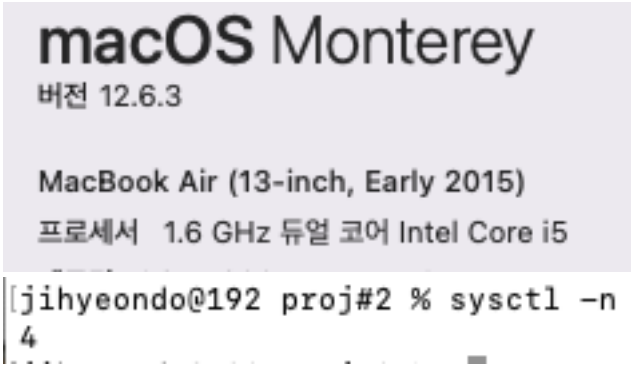
오류없이 정상적으로 컴파일되는 모습이다.

*실행 결과물

```
[jihyeondo@192 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
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 = 1부터 67108864까지 소수를 세는 중 ..........................................
.....................소수 개수 : 3957809개 .....PASSED
계산 시간 : 295.2966초
jihyeondo@192 proj#3 %
```

실행 결과가 예상 출력과 모두 일치하고 오류없이 모두 통과되는 모습을 볼 수 있다. 아마 모든 함수가 정상적으로 작동

하는 것 같다. 그런데 이번에도 교수님이 주신 예상결과와 저의 결과가 실행시간에서 꽤 많은 차이가 나는 것으로 보이는데, 이번에는 병렬처리를 하는 만큼 노트북 성능에 의해서 실행시간이 늦어지지 않았을까 생각됩니다. 물론 코드의 최적화가 부족하여 느려진 것이겠지만, 그것 만으로 10배나 되는 시간차이가 날 것 같지는 않습니다.



*프로젝트 소감 및 부족한 점

상당히 큰 범위에서 소수를 구하는 방법 그 중에서도 효율적이고 빠르게 구하는 알고리즘을 배우게 되어 신기하였습니다.

물론 아직 코드 최적화에 실력이 부족하여 생각보다 큰 시간이 소요되어 버렸지만, 이는 제가 더 노력해야 할 부분이라고 생각합니다.

이번에 학습한 밀러라빈 방식이 확률적으로 소수일 확률이 높다는 것을 이용한다는게 처음에는 확률로 계산하는 이런 방식이 맞나 싶었지만, 상당히 큰 범위에서의 계산에서 이정도 오차는 감안할 수 있을 정도로 밀러라빈 방식이 효율적일

수 있다는 것을 깨닫게 된 것 같습니다.