2019007329 이지현 proj3

mod_add 함수

a+b mod m을 계산하는 함수이다. 오버플로우를 방지하기 위해 a-(m-b)를 계산한다. 만약 a+b >= m이라면, (a-(m-b)) % m을 return하고, 그렇지 않으면 (a+b) % m을 return한다. 계산을 더욱 빠르게 하기 위해 a값과 b값을 m으로 나눈 나머지를 a, b값으로 다시 할당한 후 return 계산을 진행하였다.

mod_sub 함수

a-b mod m을 계산하는 함수이다. a < b라면 결과가 음수가 되므로 m을 더해서 양수로 만든 값을 return한다. 만약 a < b라면 (a-b+m) % m을 return하고, 그렇지 않으면 (a-b) % m을 return한다. 계산을 더욱 빠르게 하기 위해 a값과 b값을 m으로 나눈 나머지를 a, b값으로 다시 할당한 후 return 계산을 진행하였다.

mod_mul 함수

a*b mod m을 계산하는 함수이다. a*b에서 오버플로우를 방지하기 위해 덧셈을 사용하여 빠르게 계산할 수 있는 double addition 알고리즘을 사용하였다.

mod_pow 함수

a^b mod m을 계산하는 함수이다. a^b에서 오버플로우를 방지하기 위해 곱셈을 사용하여 빠르게 계산할 수 있는 square multiplication 알고리즘을 사용하였다.

find_prime 함수

miller_rabin 함수에서 (a[i]^q)^(2^j) mod n을 계산하기 위한 함수이다. 총 k번 반복하며 만약 (a[i]^q)^(2^j) == n-1 라면, 1(prime)을 return하고 k번을 반복해서 계산해도 (a[i]^q)^(2^j) == n-1 이 되지 않는다면 반복문을 빠져나와 0을 return한다. temp 변수에 할당할 (a[i]^q)^(2^j) mod n계산은 (a[i]^q) mod n, (a[i]^2q) mod n, (a[i]^4q) mod n, (a[i]^8q) mod n, ... 순서로 증가한다. 이를 통해 이전 temp 값을 두번 곱하면(제곱하면) 그 다음 temp값이 된다는 것을 알 수 있다. mod 계산에서는 a, b 각각에 먼저 mod를 적용시킨 뒤 전체 mod 연산을 진행할 수 있기 때문에 mod_mul 함수를 이용하여 temp값을 계산해주었다.

miller_rabin 함수

계산에 들어가기 앞서 n이 0이나 1일 경우, 또는 n이 2보다 큰 짝수일 경우 소수가 아니기 때문에 if문을 통해 n 값이 앞의 조건일 경우 COMPOSITE을 return한다.

먼저 (n-1) = 2^k*q를 만족하는 k, q 값을 찾는다. q는 홀수여야 하므로 q 값이 홀수가 되면 반복문을 중단한다. n-1은 무조건 짝수이기 때문에(앞 if문 조건에 의해 n은 홀수) n-1을 홀수가 될 때까지 2로 나눈다면 그 홀수 값은 q가 되고, 몇 번 나누었는가를 count 한 값은 k가 될 것이다. 초기 q 값을 n-1로 하여 q가 홀수가 될때까지 2로 나눈 후 k를 1씩 증가시킨다.

결정적 밀러 라빈 알고리즘이므로 for문은 총 12번(리스트 a의 길이) 반복한다. 먼저 a는 1보다 크고 n-1보다 작은 값이여야 한다.

1) 만약 a[i]가 n-1보다 같거나 크면 다음 계산을 생략하기 위해 continue를 사용하였다.

mod_mul 을 이용해 (a[i]^q) mod n 값을 계산해 준 후 temp 에 할당한다.

2) 만약 temp == 1 이라면 prime 이므로 다음 계산을 생략한다.

3) 만약 find_prime(n, k, temp) == 1 이라면 find_prime 함수의 return 값이 prime(1) 이라는 의미이므로 다음 계산을 생략한다.

위의 세 단계에 모두 해당되지 않아 continue 가 실행되지 않았다면 소수가 아니므로 COMPOSITE 를 return 한다.

12 개의 a 값을 모두 검사하였을때 COMPOSITE 이 reuturn 되지 않았다면 12 번의 검사 모두 PRIME 이라는 의미이므로 for 문 도중 return 되지 않고 빠져나왔다면 PRIME 을 return 한다.

컴파일

```
lejinhy@DESKTOP-IPANOF9:/mnt/c/Users/82103/Downloads/proj#3-1$ make
gcc -Wall -O3 -fopenmp -c miller_rabin.c
gcc -o test test.o miller_rabin.o  -fopenmp
```

실행 결과

```
lejinhy@DESKTOP-IPANOF9:/mnt/c/Users/82103/Downloads/proj#3-1$ ./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: 583.0339초
```

설명&소감

결정적 밀러라빈 알고리즘을 구현하였다.

수업시간에 배운 확률적 밀러라빈 알고리즘을 참고하여 구현하였는데, 결정적 밀러라빈으로 바뀌면서 새롭게 고려해야 하는 내용들을 빠르게 떠올리지 못해 아쉬웠다. 예를 들면 a 값을 select하지 않고 주어진 a 값을 검사하기 때문에 a 값이 n-1 이상일 경우를 제외해야 했고, n 값이 2보다 큰 짝수이거나 2보다 작을 경우 또한 미리 걸러낸다면 더욱 빠르게 계산할 수 있었다. 이렇게 단순하지만 중요한 조건을 인지하지 못한 채 주요 알고리즘만을 고려했던 것이 코드를 구현하는데 많은 시간을 소요하게 만들었다. 알고리즘을 구현할 때 가장 중요한 것은 기본 조건이라는 것을 깨달을 수 있었던 과제였다.

문제점

계산 시간이 500~600초대로 느린 편이다. 이번 과제는 알고리즘을 수업시간에 수도코드 형식으로 미리 배웠었기에 코드 구현도 중요하지만, 코드 알고리즘이 얼마나 좋은 성능을 보이는지가 훨씬 더 중요하다고 생각하였다. 그렇기에 내 예상보다 느린 계산 속도가 나와 매우 아쉽다. 내가 생각했을 때 계산 속도를 빠르게 만들어줄 수 있는 조건문을 모두 작성했기에 다른 친구들은 어떤 방식으로 효율적인 코드를 작성하였는지 굉장히 궁금하다.