NHẬP MÔN MÃ HÓA MẬT MÃ

BÀI TẬP NHÓM 1

I. Lý thuyết

1. Tìm hiểu

Phép kiểm tra tính nguyên tố của Fermat

- Kiểm tra

  N: là số kiểm tra

  Chọn ngẫu nhiên 1 số a từ 2 đến N-1

  Tìm ước chung lớn nhất của a và N

  Nếu khác 1 thì trả về Hợp số

  Tiếp tục kiểm tra Fermat theo công thức lý thuyết $a^{N-1} \equiv 1 \mod N$ thì trả về Số nguyên tố còn khác 1 thì trả về Hợp số

- Độ phức tạp: $O(\log^3 N)$
- Thuật toán dựa trên định lý nhỏ của Fermat $a^{N-1} \equiv 1 \mod N$

Phép kiểm tra tính nguyên tố của Miller-Rabin

- Kiểm tra:

  N: là số kiểm tra

  m: là số lẻ

  Chọn ngẫu nhiên 1 số a từ 1 đến N-1

  Tìm ước chung lớn nhất của a và N

  Nếu khác 1 thì trả về Hợp số

  Tiếp tục kiểm tra Fermat theo công thức lý thuyết $a^{N-1} \% N \neq 1$ thì trả về Hợp số

  Tiếp tục kiểm tra Miller Rabin theo công thức $N-1 = 2^k m$ và $a^m \equiv 1 \mod N$

- Độ phức tạp: $O(\log^3 N)$

- Thuật toán dựa trên định lý nhỏ của Fermat và Miller Rabin

2. Độ phức tạp và tính đúng đắn của 2 thuật toán trên

- Độ phức tạp: như nhau
- Tính đúng đắn:
  - Xác suất sai của Fermat là Số giả Fermat
  - Xác suất sai của Miller Rabin là (lnN -2)/lnN

II. Thực hành

1. Đánh giá

- Ưu điểm:
  - Dễ cài đặt.
- Nhược điểm:
  - Độ chính xác của Fermat thì thấp hơn của Miller-Rabin.
  - Độ phức tạp thời gian cao.
- Thời gian thực hiện: $a^{N-1} \equiv 1 \mod N$
  - Thời gian để sinh chuỗi s: O(len) (len: chiều dài của khóa)
  - Thời gian để chuyển từ số nhị phân sang thập phân(BigInt): **O(len * (n + log(m)))**
    len: chiều dài của khóa
    n(thời gian thực hiện phép nhân 2 số BigInt): chiều dài của chuỗi 2^x
    log(m)(thời gian thực hiện phép mũ)
  - Thời gian để tính $a^{N-1} \mod N$ (hàm binpow): O(2n.log(n))
    n: chiều dài của số đang kiểm tra
    2n: thời gian để thực hiện phép nhân và module

```
bool fermat_testing(BigInt N) {
    BigInt temp;
    string s;
    s = generateBinaryString(Length(N) - 1);
    BigInt a;
    a = StringtoBigInt(s);

    temp = N - 1;
    if (binpow(a, temp, N) == 1)
        return true;
    else
        return false;
    return true;
}
```

⇨ Thời gian thực hiện của thuật toán: **O(len + (len * (n + log(m)) + (2n.log(n)))**

2. Báo cáo chi tiết

a) Cách chạy chương trình:

- Đầu tiên, màn hình console hiện lên yêu cầu người dùng nhập độ dài khóa.
- Tiếp theo, người dùng nhập khóa rồi Enter để chạy chương trình.
- Sau đó chờ chương trình chạy cho tới khi in ra được khóa trên màn hình console.

b) Thiết kế chương trình:

- Tính toán và lưu trữ số nguyên lớn
  - string digits: dùng để lưu trữ số nguyên lớn
  - Overload các hàm operator để tính toán các số nguyên lớn

```
class BigInt {
    string digits;
public:
    BigInt(unsigned long long n = 0);
    BigInt(string&);
    BigInt(BigInt&);

    friend void divide_by_2(BigInt& a);
    friend bool Null(const BigInt&);
    friend int Length(const BigInt&);
    int operator[](const int)const;

    friend BigInt operator%(const BigInt&, const BigInt&);
    friend BigInt& operator%=(BigInt&, const BigInt&);

    friend BigInt& operator*=(BigInt&, const BigInt&);
    friend BigInt operator*(const BigInt&, const BigInt&);
```

- Các sinh số nguyên tố lớn tương ứng với từng độ dài khóa
  - Sử dụng hàm generateBinaryString để sinh chuỗi bit tương ứng với độ dài khóa (set bit đầu và cuối bằng 1)
  - Sử dụng hàm StringtoBigInt để chuyển từ chuỗi bit sang BigInt

```
BigInt StringtoBigInt(string binary) { ... }

string generateBinaryString(int N)
{
    srand(time(NULL));
    string S = "";
    S += '1';
    N -= 2;

    for (int i = 0; i < N; i++) {
        int x = ((int)rand() % 2);

        S += to_string(x);
    }

    S += '1';
    return S;
}
```

- Sử dụng hàm fermat_testing để kiểm tra số vừa sinh có phải là số nguyên tố hay không?

c) Đánh giá:

- Ưu điểm:
  - Dễ cài đặt.
- Nhược điểm:
  - Độ phức tạp thời gian cao.

- Thời gian thực hiện: O(a.(**len + (len * (n + log(m)) + (2n.log(n))**))
  a: số lần sinh khóa để được số nguyên tố