ReportLab1

Alessandro Ravveduto: 1020539
Francesco Testa: 1029406
Mattia Lodi: 1020617

09 Aprile 2024

1 Implementazione Algoritmi C

Nel riportare le implementazioni è stato omesso il main. Ogni programma controlla il numero dei parametri per poi chiamare la funzione corrispondente. Per la visualizzazione dei risultati basti vedere l'output sul terminale.
Poiché in C l'operatore % è il resto e non il modulo, è stata implementata una semplice funzione che esegue il modulo Euclideo.

```
int modulo(int a, int b){
    int m = a % b;
    if(m < 0)
        m = (b < 0) ? m - b : m + b;
    return m;
}
```

La differenza tra i due operatori si capisce molto facilmente con un esempio:

- Resto: -21 % 4 = -1 perchè -21 / 4 = -5 e -21 - (-20) = -1
- Modulo: -21 mod 4 = 3 perchè -21 + 4 x 6 = 3

1.1 Euclidean Algorithm

```
struct listOfNumbers
{
    int number;
    struct listOfNumbers* next;
};

struct listOfNumbers* addNumber(struct listOfNumbers* head, int number) {
    struct listOfNumbers* newNum = malloc(sizeof(struct listOfNumbers));
    if (newNum == NULL) {
        perror("malloc failed\n");
        exit(EXIT_FAILURE);
    };
    newNum->next = NULL;
```

```c
    newNum->number = number;

    if (head == NULL) {
        head = newNum;
    } else {
        struct listOfNumbers* tmp = head;
        while (tmp->next != NULL) {
            tmp = tmp->next;
        }
        tmp->next = newNum;
    }
    return head;
}

void printQuotients(struct listOfNumbers* head) {
    struct listOfNumbers* current = head;
    printf("Quotients (from 1 to m):\n");
    int i=1;
    while (current != NULL) {
        printf("q%d: %d\n", i, current->number);
        current = current->next;
        i++;
    }
}

void euclideanAlg(int a, int b) {

    struct listOfNumbers* allQuotients = NULL;

    int qCurrent;
    int rPrec = a;
    int rCurrent = b;
    int rNew;

    while(rCurrent != 0) {
        qCurrent = floor(rPrec/rCurrent);
        rNew = rPrec - qCurrent*rCurrent;
        allQuotients = addNumber(allQuotients, qCurrent);
        rPrec = rCurrent;
        rCurrent = rNew;
    }

    printQuotients(allQuotients);
    printf("r: %d\n", rPrec);
}
```

Per la gestione della lista di q da ritornare è stato necessario implementare una struct che la gestisca. L'addNumber inserisce un nodo in coda, mentre il printQuotients serve per stampare tutti i quozienti.

1.2 Extended Euclidean Algorithm

```c
//result[3] = r,s,t
void euclideanAlgExt(int a, int b, int* result) {
    int a0 = a;
    int b0 = b;
```

```c
    int t0 = 0;
    result[2] = 1;
    int s0 = 1;
    result[1] = 0;
    int q = floor(a0/b0);
    result[0] = a0 - q*b0;
    int temp;

    while (result[0]>0) {
        temp = t0 - q*result[2];
        t0 = result[2];
        result[2] = temp;
        temp = s0 - q*result[1];
        s0 = result[1];
        result[1] = temp;
        a0 = b0;
        b0 = result[0];
        q = floor(a0/b0);
        result[0] = a0 - q*b0;
    }
    result[0] = b0;
    printf("gcd:(%d,%d) = %d*%d+%d*%d = %d\n", a, b, result[1], a, result[2], b, result[0]);
}
```

Nel main è inizializzato un array di 3 elementi e passato alla funzione.

1.3 Multiplicative Inverse

```c
int multliplicativeInverse(int a, int b) {
    int a0 = a;
    int b0 = b;
    int t0 = 0;
    int t = 1;
    int q = (int)(a0/b0);
    int r = a0 - (q*b0);
    int temp;

    while (r>0) {
        temp = modulo((t0 - (q*t)), a);
        t0 = t;
        t = temp;
        a0 = b0;
        b0 = r;
        q = (int)(a0/b0);
        r = a0 - q*b0;
    }

    if (b0 != 1) {
        printf("No multiplicative inverse\n");
        return -1;
    }

    printf("Multiplicative inverse of %i mod %i is %d\n", b, a, t);
    return t;
}
```

1.4 Square and Multiply

```c
#define IS_BIT_SET(n,x) ((n & (1 << x)))

unsigned int binaryLenght(int c) {
    unsigned int bits;
    unsigned var = (c<0) ? -c : c;
    for (bits = 0; var!=0; bits++) var >>= 1;
    return bits;
}

int squareNmultiply(int x, int c, int n) {
    int z = 1;

    for (int i = binaryLenght(c)-1; i >= 0  ; i--) {
        z = (z * z) % n;
        if (IS_BIT_SET(c, i)) {
            z = modulo(z*x, n);
        }
    }
    printf("%d^%d mod %d = %d\n", x, c, n, z);
    return z;
}
```

Per calcolare la lunghezza ℓ dell'esponente è stata implementata una funzione ausiliaria, mentre per accedere al singolo bit nell'iterazione sono stati utilizzati gli operatori bitwise.
A partire da questi calcoli è stato implementato l'algoritmo presente sulle slide.

2 test

Per compilare è necessario eseguire `make`.
Per eliminare i compilati si può eseguire `make clean`.

2.1 Esercizio 1

Compute gcd(57, 93) using the Euclidean algorithm, and find integers s and t such that 57s + 93t = gcd(57, 93).

```
$ ./Gcd 57 93
Quotients (from 1 to m):
q1: 0
q2: 1
q3: 1
q4: 1
q5: 1
q6: 2
q7: 2
r: 3
$ ./ExtendedEuclideAlg 57 93
gcd(57,93) = -13*57+8*93 = 3
```

2.2 Esercizio 2

Compute the following multiplicative inverses:

- $17^{-1} \, mod \, 101$

```
$ ./MultiplicativeInverseAlg 17 101
Multiplicative inverse of 17 mod 101 is 6
```

- $357^{-1} \, mod \, 1234$

```
$ ./MultiplicativeInverseAlg 357 1234
Multiplicative inverse of 357 mod 1234 is 1075
```

- $3125^{-1} \, mod \, 9987$

```
$ ./MultiplicativeInverseAlg 3125 9987
Multiplicative inverse of 3125 mod 9987 is 1844
```

2.3 Esercizio 3

Suppose Bob chooses p = 101 and q = 113. Then n = 11413 and ϕ(n) = 100 $\times$ 112 = 11200. Since 11200 = 26527, an integer b can be used as an encryption exponent if and only if b is not divisible by 2, 5, or 7.
Suppose Bob chooses b = 3533.

- Please verify that gcd(ϕ(n), b) = 1 using the Euclidean Algo.

```
$ ./Gcd 11200 3533
Quotients (from 1 to m):
q1: 3
q2: 5
q3: 1
q4: 7
q5: 4
q6: 3
q7: 2
q8: 2
r: 1
```

- Now compute Bob's secret decryption exponent, a, using the Multiplicative Inverse Algorithm

```
$ ./MultiplicativeInverseAlg 11413 3533
Multiplicative inverse of 11413 mod 3533 is 1454
```

Bob publishes n = 11413 and b = 3533 in a directory.
Now, suppose Alice wants to encrypt the plaintext 9726 to send to Bob. She will compute $9726^{3533} \, mod \, 11413$ and send the ciphertext c over the channel.

- Please determine c's value using the square and multiply algorithm

```
$ ./SquareAndMultiply 9726 3533 11413
9726^3533 mod 11413 = 5761
```

Il ciphertext c quindi è 5761.

When Bob receives the ciphertext c, he uses his secret decryption exponent a to compute the plaintext sent by Alice.