Trabalho 8

Vítor Amorim Fróis - 12543440

Parte 1

Para o exercício 4, foi escolhida a recorrência linear homogênea de segundo grau

$a_n = 2a_{n-1} + 3a_{n-2}$ e o termo $g_1(n) = n$.

Todas as soluções possíveis das recorrências não homogêneas da soma dos termos anteriores pode ser expressa pela **combinação linear** $X_n = P(n) + A(r')^n + B(r'')^n$, onde $P(n)$ é um polinômio de grau 1 que representa uma forma genérica de representar a_n.

Primeiro deve-se encontrar as raízes de a_n

$a_n = 2a_{n-1} + 3a_{n-2}$

Toda recorrência pode ser expresso através de uma equação quadrática, onde cada coeficiente é um termo. Assim:

$x^2 - 3x - 2 \leftarrow$ Aplica Bháskara e encontra as raízes $\frac{3 \pm \sqrt{17}}{2}$

Todo termo a_n pode ser escrito através de $a_n = P(n) = in + j$

Portanto, $in + j = 3(in - i + j) + 2(in - 2i + j) + n$

Simplificando, $0 = 4in - 7i + 4j + n$

Resolvendo o sistema chutando valores de n, obtém-se $i = \frac{-1}{4}$ e $j = \frac{-7}{16}$.

Logo, $P(n) = \frac{-1}{4}n + \frac{-7}{16}$. Agora basta montar a equação final através de combinação linear
$X_n = P(n) + A(r')^n + B(r'')^n$

$$X_n = \frac{-1}{4}n + \frac{-7}{16} + A(\frac{3 + \sqrt{17}}{2})^n + B(\frac{3 - \sqrt{17}}{2})^n \qquad \forall A, B \in \mathbb{R}$$

Parte 2

Para implementar o sistema RSA em C, é necessário utilizar alguns códigos implementados anteriormente, como o Extendido de Euclides, que retorna o MCD de dois números e possibilita calcular o inverso de um número em $\mathbb{Z}_d$.

```
/**
 * @brief O algoritmo estendido de euclides (AEE) realiza\n
 * operações MDC/GCD sucessivas até encontrar gcd = 1,
 * isto é, quando b%a=0.
 * @param a termo 1
 * @param b termo 2
 * @param x variável para armazenar coeficientes
 * @param y variável para armazenar coeficientes
```

```
 * @return o resultado do algoritmo estendido de euclides entre a e b
 */
llint euclides(llint a, llint b, llint* x, llint* y){
    // Considere a equação
    // (a*x)+(b*y)=1
    // a e b são divisores do número A em AX⁻¹≃(mod m)
    // com m sendo o módulo passado.
    // Quando o AEE=1 e a=0, os valores de x e y para
    // encontrar a equação podem ser obtidos recursivamente
    // sabendo que:
    // --> novo x = y - (b / a) * x
    // --> novo y = x

    if(a == 0){
        *x = 0;
        // printf("x = 0\n");
        *y = 1;
        return b;
    }

    llint newX, newY;
    //chamadas recursivas do algoritmo com novos valores
    int gcd = euclides(b%a, a, &newX, &newY);

    //encontra e retorna por referência novos valores de x e y
    *x = newY - (b/a)*newX;
    // printf("x = %lld - (%lld/%lld) * %lld = %lld\n", newY, b, a, newX, *x);
    *y = newX;
    // printf("y = %lld\n", *y);
    // printf("(%lld*%lld)+(%lld*%lld)=1\n\n", a, *x, b, *y);

    return gcd;
}

/**
 * @brief Calcula o inverso de a em Zm baseado no Algoritmo Estendido de Euclides
 *
 * @param a
 * @param m
 */
llint inverso(llint a, llint m){
    llint x, y;
    int mdc = euclides(a, m, &x, &y);
    if(mdc != 1){
        // printf("\n\nO inverso de %lld em %lldZ é inexistente,\n", a, m);
        // printf("dado que os números não são primos entre si.\n");
        // printf("AEE(%lld, %lld) = %d", a, m, mdc);
        return -1;
    }else{
        //É feito um tratamento para casos em que o x é negativo,
        //considerando que o sistema Zm é circular.
        x = (x % m + m)%m;
        //printf("\n\nO inverso de %lld em %lldZ é %lld.", a, m, x);
        return x;
    }
}
```

Além disso, foi necessário aprimorar a função para cálculo de módulo implementada no trabalho anterior

```
/**
 * @brief Calcula a^r(mod m)
 */
llint calc_modulo(llint a, llint r, llint m){
    llint result = 1;
    //Checa se a é divisível por m
```

```
    a = a % m;

    //Se a for divisível, o resto é zero e não há
    //necessidade de fazer contas
    if(a == 0) return 0;

    //Calcula o módulo através da divisão sucessiva de r por 2
    //Assim, a eficiência do algoritmo é maior por não precisar
    //Loopar vários for
    while(r > 0){
        //Caso r for ímpar, execute
        if(r % 2 != 0)
            result = (result * a) % m;

        //Com o print, é possível ver o quão eficiente a função é.
        //Geralmente não foram gastos mais que 10 iterações do while
        //para resolver o problema.
        printf("a=%u r=%u m=%u res=%u\n", a, r, m, result);
        a = (a * a) % m;
        r = r / 2;
    }
    return result;
}
```

Por fim basta implementar o sistema RSA na main(). Os primos são escolhidos manualmente, tal que $n = p \times q$ seja menor que 2^{16}. Para codificar a mensagem m, é necessário calcular $m^e \mod n$, com o e escolhido a mão, justificado no código e n sendo o produto dos primos.

Já para decodificar, é necessário encontrar f, e aí se encontra a segurança do método. Para o cálculo de f é necessário $\phi = (p-1) * (q-1)$. Como é muito pesado fatorar um número como n computacionalmente, consequentemente é muito difícil encontrar phi e por fim, f, que é o inverso de e em $\mathbb{Z}_\phi$.

🔴 O código está comentado!

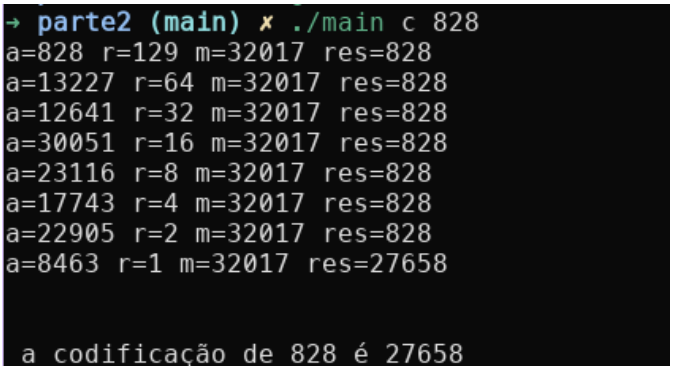
```
int main(int argc, char *argv[]){
    //Tratamento da entrada
    char input = argv[1][0];
    llint mensagem = (llint) atoll(argv[2]);
    if(argc != 3){
        printf("'c' para codificar\n");
        printf("'d' para decodificar\n");
        printf("./main x xxxx\n");
        return 0;
    }

    //Dados do rsa
    Dados rsa;
    //p e q primos > 100 escolhidos a mão
    rsa.p = 101;
    rsa.q = 317;
    rsa.n = rsa.p * rsa.q;
    rsa.phi = (rsa.p-1)*(rsa.q-1);
    rsa.e = 129;
    //Para calcular o módulo, a cada iteração do while, caso ele seja ímpar,
    //mais operações devem ser executadas. Assim, é bom escolher um número da
    //forma 10000000001. A cada divisão por 2, ele será sucessivamente par.
    rsa.f = inverso(rsa.e, rsa.phi);

    //Para codificar, roda a função calc_modulo com e escolhido;
    if(input == 'c') printf("\n\n a codificação de %u é %u\n", mensagem, calc_modulo(mensagem, rsa.e, rsa.n));
```

```
    //Para decodificar, roda com o inverso de e em Zphi, calculado com a função inverso();
    else if(input == 'd') printf("\n\n a decodificação de %u é %u\n", mensagem, calc_modulo(mensagem, rsa.f, rsa.n));
    return 0;
}
```

Para as entradas pedidas pelo professor, obtém-se:



Codificação

Decodificação