UNIVERSIDADE DE BRASÍLIA
Campus UnB Gama (FGA)
Curso: Engenharia de Software
Professor: Cristiane Loesch de Souza Costa **Data:** 15/10/2025
Disciplina: Matemática Discreta II **Turma:** T01

ATIVIDADE PARA NOTA
(Implementação em C ou C++)

INFORMAÇÕES GERAIS:

DATA DA ENTREGA: até 18/10/2025

HORÁRIO LIMITE: até o meio-dia

FORMATO DA ENTREGA: O link do github dos programas e documento com as respostas da questão 4 deverão ser enviados via Teams (favor aceitar convite de envio a sessão caso necessário seja, fique atento a seu email)

NOTA: até 2,0 pontos na avaliação P2

A atividade é composta por 4 questões de desenvolvimento de código e uma delas, a questão 4, possui parte com respostas V ou F e conclusões sobre a saída do código.

Os alunos poderão realizar os códigos individualmente ou em dupla.

O cabeçalho do código deverá conter nome do aluno e matrícula.

Após o horário limite de envio nenhum código poderá ser alterado, implicando em risco de a questão ser anulada pelo professor.

Os códigos deverão funcionar sem limitações, exceto as previamente definidas, e serão testados e lidos pelo professor para garantir que todos os passos exigidos tenham sido contemplados no desenvolvimento.

Lembre-se de respeitar as exigências de cada uma das questões e de ler o documento em detalhes, até o final, para evitar problemas futuros.

QUESTÃO 01 (1,0 ponto)

Projeto de Programação: Sistema RSA com Fatoração ρ de Pollard e Aplicação de Teoremas Modulares em Três Etapas.

Objetivo:
Implementar em C ou C++ um sistema completo de criptografia e descriptografia RSA, iniciando pela fatoração de números compostos usando o método ρ de Pollard, e aplicando corretamente conceitos de aritmética modular (como o Teorema de Fermat, o Teorema de Euler e a Divisão Euclidiana) para os cálculos de potência modular durante a codificação e decodificação de mensagens.

Etapa 1: Fatoração Interativa (Método ρ de Pollard)

Objetivo: Descobrir os fatores primos p e q de dois números compostos N_1 e N_2.

Entrada de dados:

O programa deve solicitar **dois números compostos distintos** N_1 e N_2.

Restrição: Cada número deve possuir **3 ou 4** dígitos, ou seja, entre 100 e 9999.

Informe ao usuário que cada N_i deve ser produto de **primos distintos** para que o método ρ de Pollard seja eficiente.

Implementação do método ρ de Pollard:

Utilize a função de iteração: $g(x)=(x^2+1) \bmod N_i$

Semente – $x_0=2$.

Em cada iteração, calcule: mdc ($|x_2-x_1|$,N_i) até encontrar um fator p_i não trivial de N_i

O programa deve exibir **cada passo da iteração**.

Definição dos primos RSA:

Seja p o fator encontrado de N_1

Seja q o fator encontrado de N_2.

Exiba claramente os valores de p e q.

****Observação:** O cálculo do mdc deve ser feito **utilizando o Algoritmo de Euclides**, implementado pelo aluno (não é permitido usar funções prontas como std::gcd).

Etapa 2: -Geração das Chaves RSA

Objetivo: Construir o par de chaves pública e privada do sistema RSA.

Cálculo do módulo: $n = p \times q$

Totiente de Euler: $z(n) = (p-1) \times (q-1)$

Escolha do expoente público: Escolha o menor E >1 e E< n tal que mdc (E , z(n))=1

Cálculo do expoente privado: Encontre D tal que: $D \times E \equiv 1 \bmod z$

*Utilize o **Algoritmo Estendido de Euclides** para determinar o inverso modular de E em relação a z

Impressão das chaves:

Chave pública: (n,e)

Chave privada: (n,d)

Etapa 3 - Codificação (Criptografia) e Decodificação (Descriptografia)

Objetivo: Realizar a criptografia e a decodificação de uma mensagem, aplicando o teorema modular adequado e um sistema próprio de codificação numérica de letras.

Pré – Codificação

Antes de aplicar a criptografia RSA, cada caractere da mensagem deve ser convertido em um número segundo o sistema de pré-codificação do alfabeto: A = 11, B= 12, …, Z= 36. Espaço = 00.

Codificação

Para cada bloco M formado pelos números da mensagem: $C \equiv M^E \pmod{n}$

O programa deve exibir **o cálculo passo a passo** da exponenciação modular.

Decodificação

Para cada bloco cifrado C: $M \equiv C^D \pmod{n}$

O resultado M deve ser reconvertido para letras segundo a tabela de pré-codificação.

* Lembre-se cada bloco será referente a apenas 2 dígitos.

Resolução da exponenciação modular

Durante o cálculo de $M^E \pmod{n}$ e $C^D \pmod{n}$, o programa deve:

- Verificar as condições e selecionar automaticamente o método de redução de expoente:
    - **Pequeno Teorema de Fermat**, se n for primo;
    - **Teorema de Euler**, se mdc(M,n)=1;
    - **Teorema da Divisão Euclidiana**, para reduzir o expoente.
- O programa deve indicar na saída textual qual teorema foi aplicado e mostrar o cálculo correspondente.

Observações

- Espaços e pontuações podem ser ignorados ou substituídos por um código fixo (exemplo: 00 para espaço).
- O código deve ser implementado **em C ou C++**, sem uso de bibliotecas externas de criptografia.
- Todas as funções fundamentais (como cálculo de mdc, inverso modular, e exponenciação modular) devem ser **programadas pelo aluno**.
- O programa deve **imprimir o passo a passo de todos os pontos principais do cálculo**, incluindo:
    1. Iterações do método ρ de Pollard;
    2. Cálculo do mdc (Algoritmo de Euclides);
    3. Determinação do inverso modular (Euclides Estendido);
    4. Escolha e aplicação do teorema modular (Fermat, Euler ou Divisão Euclidiana);
    5. Processo completo de criptografia e descriptografia;
    6. Reconversão numérica em texto.
- O sistema deve confirmar que a mensagem decifrada é idêntica à mensagem original.
- Os alunos devem comentar no código as decisões tomadas e justificar o método modular escolhido em cada etapa.

QUESTÃO 02 (0,5 ponto)

“Chaves Periódicas”

Resolva o problema abaixo, inspirado no problema das “Cigarras Periódicas” da Maratona de Programação da SBC 2017 (Fase Regional, URI Online Judge / Beecrowd, Problema 2660).

Em um laboratório de criptografia experimental, várias chaves periódicas possuem ciclos de ativação em anos. Cada chave K_i só pode ser utilizada em anos múltiplos do seu ciclo C_i.

Todas as chaves foram **ativadas simultaneamente no ano 0**.

Você recebe uma lista com **N ciclos** de chaves: $C_1, C_2, ..., C_N$.

Objetivo: Descobrir o primeiro ano futuro (maior que 0) em que todas as chaves podem ser utilizadas simultaneamente.

Limite de Ano : Considere apenas anos de 1 a 50.

- Se não houver nenhum ano dentro deste intervalo em que todas as chaves possam ser usadas simultaneamente, o programa deve informar ao usuário que não é possível.
- Caso exista, o programa deve imprimir o primeiro ano sincronizado dentro do limite.

Entrada

- A primeira linha contém um inteiro N ($1 \leq N \leq 10$), o número de chaves.
- A segunda linha contém N inteiros $C_1, C_2, ..., C_N$ ($2 \leq C_i \leq 20$), representando os ciclos de ativação das chaves.

Saída

- Um único inteiro: o **primeiro ano X>0** em que todas as chaves podem ser utilizadas simultaneamente **dentro do limite de 50 anos**.
- Caso não exista ano válido, exiba uma mensagem informando impossibilidade.

Observação para Pesquisa

- Cada chave ativa-se apenas em múltiplos do seu ciclo.
- Para resolver o problema, é recomendado pesquisar o problema das cigarras periódicas para entender como **sincronizar eventos periódicos com ciclos diferentes**.
- O enunciado não especifica **qual método utilizar**, deixando para o aluno investigar e decidir como implementar a solução.

QUESTÃO 03 (0,5 ponto)

“A Razão de Eficiência de um Número”

Inspirado em recursos utilizados na solução de problemas de maratonas de programação.

Enunciado

Na Teoria dos Números, algumas funções descrevem propriedades fundamentais de um inteiro N. Duas delas são:

1. **Função τ(N)** — conta o **número total de divisores de N**.
2. **Função σ(N)** — calcula a **soma de todos os divisores de N**.

A **Razão de Eficiência** de um número N é definida como:

$$\text{Razão de Eficiência}(N) = \frac{\sigma(N)}{\tau(N)}$$

Seu objetivo é **calcular a Razão de Eficiência de N** e imprimir o resultado com **duas casas decimais de precisão**.

Entrada → Um único inteiro N (1 ≤ N ≤ 105).

Saída → Um número real: a Razão de Eficiência de N com duas casas decimais.

Observações:

→ **Fatoração Prima**

- Decomponha N em seus fatores primos: $N = p_1^{a_1} \cdot p_2^{a_2} \cdot \cdots \cdot p_r^{a_r}$
- Este é o passo **fundamental**, pois todas as fórmulas seguintes dependem dessa decomposição.
- **Dica:** Para N ≤ 105, use **Trial Division (tentativa e erro)**, verificando divisores até N
    - A fatoração prima deve ser feita de forma eficiente, dividindo N pelo fator primo p o máximo de vezes possível

→ **Cálculo de τ(N) e σ(N) e da razão**

- A partir da fatoração prima, utilize as fórmulas de tau e sigma
- Calcule a razão de eficiência
    - Lembrete: a saíde é um número real (float ou double) para garantir as duas casas decimais.

→ **Passo a Passo no Código**

- Para fins de aprendizado, o programa deve imprimir os passos principais, incluindo:
    - Lista de fatores primos e seus expoentes.
    - Cálculo intermediário de τ(N).
    - Cálculo intermediário de σ(N).
    - Resultado final da Razão de Eficiência.

EXTRA: Limites de Eficiência

- Algoritmos mais avançados como **Pollard's Rho** não são necessários para N≤105.
- O método de **Trial Division** é suficiente e ensina os fundamentos de fatoração.

QUESTÃO 04 (bônus – 0,5 pontos) (Arthur Fernandes, 2025)

Com o código abaixo, complete as **linhas 10, 14, 23, 36, 45, 72 e 78** que faltam para que o programa funcione corretamente, realizando a divisão modular e o cálculo da congruência **H ÷ G (mod Zn)** seguido de **a^x mod n1**, aplicando o **Pequeno Teorema de Fermat** ou o **Teorema de Euler**, conforme o caso.

```c
1   #include <stdio.h>
2
3   #ifdef _WIN32
4   #include <windows.h>
5   #endif
6
7   // Função para calcular o máximo divisor comum (MDC) com exibição dos passos
8   int mdcComPassos(int a, int b) {
9       int resto;
10      while ([1] != 0) {
11          resto = a % b;
12          printf("Algorítmo de Euclides: %d mod %d = %d\n", a, b, resto);
13          a = b;
14          b = [2];
15      }
16      return a;
17  }
```

```c
18  int inversoModular(int a, int m) {
19      int m0 = m, t, q;
20      int x0 = 0, x1 = 1;
21      int A = a, B = m;
22
23      [3](a, m);
24
25      while (m != 0) {
26          q = a / m;
27          t = m;
28          m = a % m;
29          a = t;
30
31          t = x0;
32          x0 = x1 - q * x0;
33          x1 = t;
34      }
35      if (x1 < 0)
36          [4] += m0;
37      printf("\nSubstituindo, temos que o inverso de %d em %d é %d.\n\n", A, B, x1);
38      return x1;
39  }
```

```c
41  int powMod(int base, int exp, int mod) {
42      long long res = 1;
43      long long b = base % mod;
44      while (exp > 0) {
45          if ([5])
46              res = (res * b) % mod;
47          b = (b * b) % mod;
48          exp >>= 1;
49      }
50      return (int)res;
51  }
```

```c
int main() {
#ifdef _WIN32
    SetConsoleOutputCP(CP_UTF8);
#endif

    int H, G, Zn, x, n1;

    printf("Insira H: ");
    scanf("%d", &H);
    printf("Insira G: ");
    scanf("%d", &G);
    printf("Insira Zn: ");
    scanf("%d", &Zn);
    printf("Insira x: ");
    scanf("%d", &x);
    printf("Insira n1: ");
    scanf("%d", &n1);
    printf("\n");

    int inverso = [6](G, Zn);
    int a = (H * inverso) % Zn;

    printf("Fazendo a multiplicação modular: %d * %d mod %d ≡ %d\n", H, inverso, Zn, a);
    printf(" Sendo %d o inverso de %d.\n", inverso, G);

    int resultado = [7](a, x, n1);
    printf("Valor final da congruência: %d\n", resultado);

    return 0;
}
```

Com o código completo e preenchido corretamente, qual seria a saída com os valores: H: 7, G: 3, Zn: 11, x: 10, n1: 13

2. Considere o **código abaixo**, que realiza o cálculo da divisão modular H ÷ G (mod Zn) e depois computa a^x mod n1, aplicando o Pequeno Teorema de Fermat ou o Teorema de Euler, conforme a natureza de `n1`, classifique como Verdadeiro (V) ou Falso (F) cada uma das afirmativas a seguir:

```c
int inversoModular(int a, int m) {
    int m0 = m, t, q;
    int x0 = 0, x1 = 1;
    while (m != 0) {
        q = a / m;
        t = m;
        m = a % m;
        a = t;

        t = x0;
        x0 = x1 - q * x0;
        x1 = t;
    }
    if (x1 < 0)
        x1 += m0;
    return x1;
}
```

( ) O algoritmo de Euclides estendido é utilizado para calcular o **inverso modular** de um número.

( ) Se mdc(G, Zn) ≠ 1, o programa ainda consegue encontrar o inverso de G em Zn.

( ) A operação (H * inverso) % Zn representa a **divisão modular de H por G**.

( ) Se n1 for primo, o código aplica o **Pequeno Teorema de Fermat** para simplificar o cálculo de a^x mod n1.

( ) A função powMod implementa o cálculo de potência modular utilizando multiplicações diretas sem otimização.

( ) Quando o resultado do inverso é negativo, o código ajusta o valor somando o módulo m0.

( ) O cálculo de fi(n1) (função totiente de Euler) é utilizado apenas quando n1 **não é primo**.

Lembrete Final para Implementação (Aplicável a Todas as Questões)

1. **Passo a Passo Visível**
    - Em todas as etapas do programa, **imprima os cálculos e decisões intermediárias**.
    - Exemplos de itens a mostrar:
        - Iterações de algoritmos (como Pollard ρ ou verificação de múltiplos).
        - Cálculos de MDC e inverso modular.
        - Fatores primos e expoentes.
        - Aplicação de teoremas (Fermat, Euler, Divisão Euclidiana) com indicação clara do método usado.
        - Cálculos intermediários de τ(N), σ(N) e Razão de Eficiência.
2. **Legibilidade e Numeração**
    - Numere cada passo e inclua **legendas explicativas**, para que alguém lendo a saída entenda **como o resultado foi obtido**.
3. **Justificativa de Métodos**
    - Comente no código **por que cada método modular ou algoritmo foi escolhido**, especialmente quando há alternativas possíveis.
4. **Validação e Interatividade**
    - Solicite entradas do usuário e **valide os limites ou restrições** (como N1/N2 de 3–4 dígitos, anos de 1 a 50, etc.).
    - Caso não haja solução válida, **informe claramente ao usuário**.
5. **Integração das Técnicas**
    - Combine algoritmos e teoremas de forma coerente, mostrando **como cada técnica contribui para o resultado final**.
    - **Todos os cálculos necessários devem utilizar técnicas aprendidas na disciplina de MD2**, reforçando a aplicação prática do conteúdo da disciplina.
6. **Confirmação de Resultados**
    - Sempre que possível, compare a saída final com os dados originais (por exemplo, mensagem decifrada deve ser **idêntica à original**).