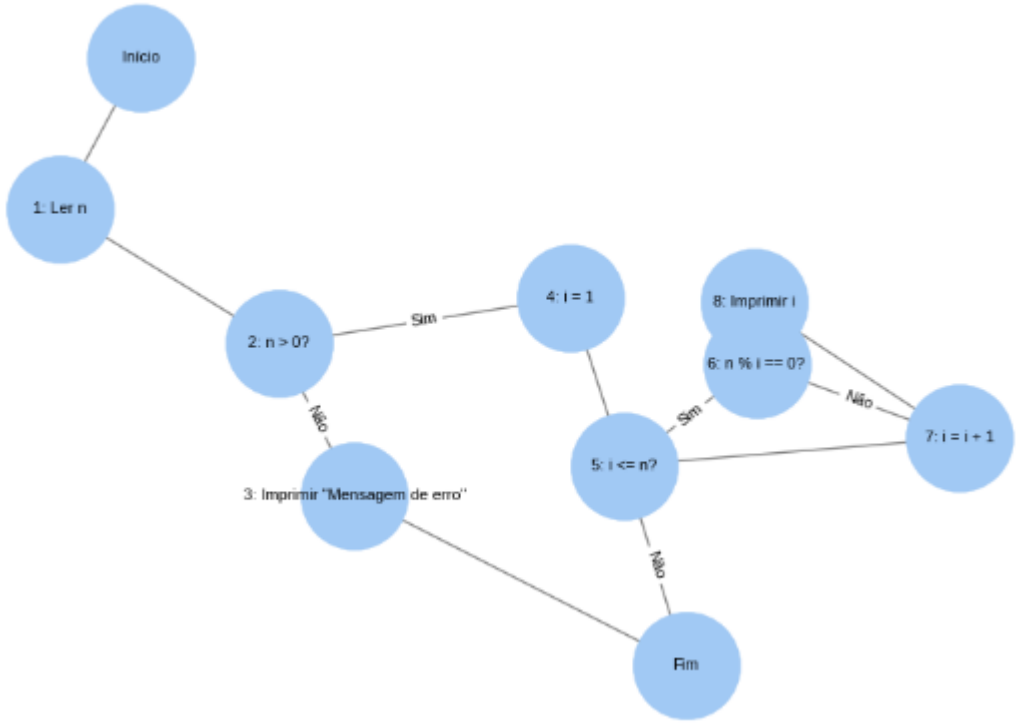
Algoritmo (a) – Divisores de um Número

Fluxo: Ler número -Testar se > 0 - Para cada i de 1 até n - Testar se n % i == 0 - Imprimir divisor - Fim.

Grafo de Fluxo:



Complexidade Ciclomática:

- **V(G) =** 3 regiões (decisão inicial + loop com condição interna).
- **V(G) =** 7 arestas - 6 nós + 2 = 3.
- **V(G) =** 2 nós predicados + 1 = 3.

Conjunto Base de Caminhos Independentes:

 Caminho 1: Entrada inválida (n <= 0) → 1 – 2 – Não – 3 - Fim.

 Caminho 2: Entrada válida, mas só divisor 1 → 1 – 2 – Sim – 4 – 5 – Sim – 6 – Sim – 8 – 7 – 5 – Não - Fim (ex: n=1, uma iteração).

 Caminho 3: Entrada válida, múltiplos divisores → 1 – 2 – Sim – 4 – 5 – Sim -6 – Sim – 8 – 7 – 5 – Sim – 6 – Não – 7 – 5 – Sim – 6 – Sim – 8 - 7- ... – 5 – Não - Fim (múltiplas iterações com condições verdadeiras/falsas).

Casos de Teste:

Caminho	Entrada	Resultado Esperado
1	n = -5	Mensagem de erro
2	n = 1	Divisores = [1]
3	n = 6	Divisores = [1, 2, 3, 6]

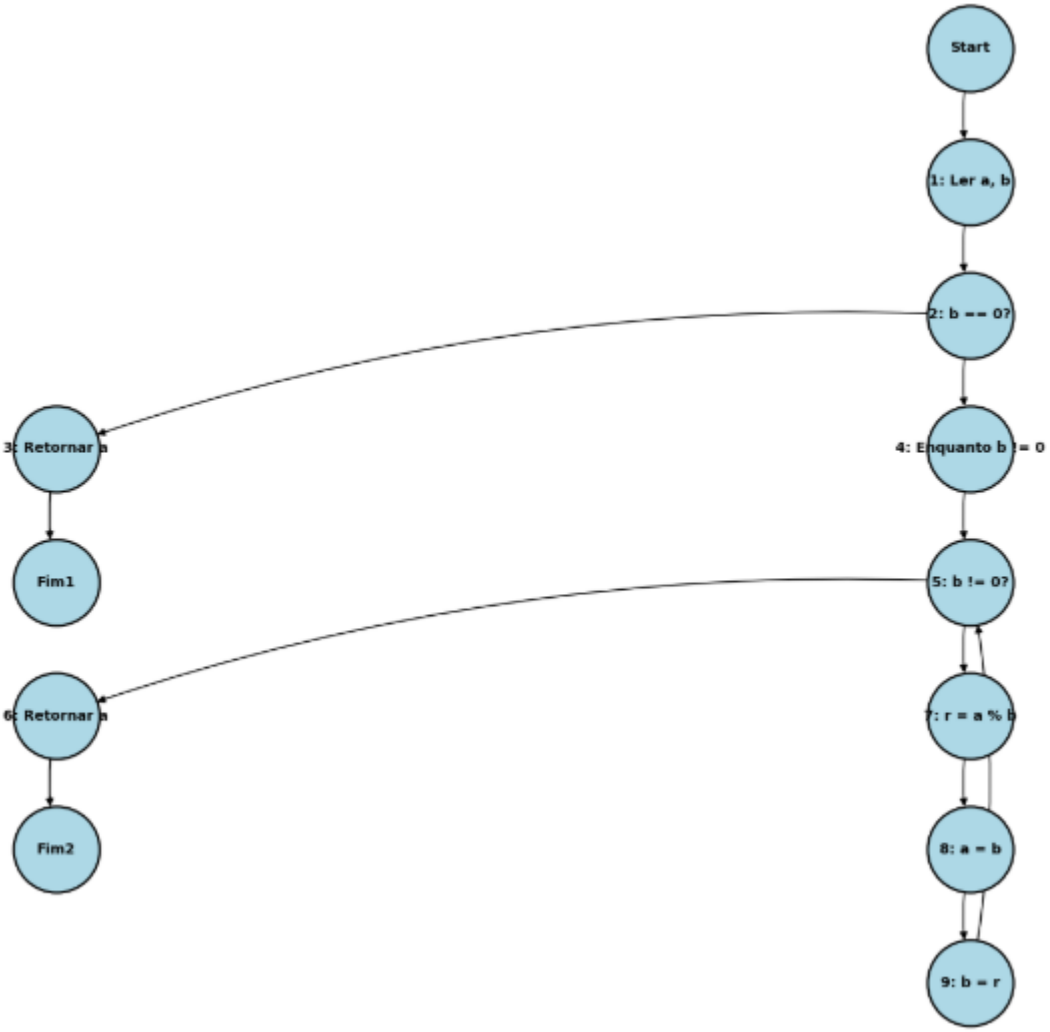
Implementação em Python:

```
def divisores(n):
    if n <= 0:
        print("Mensagem de erro")
        return
  for i in range(1, n+1):
        if n % i == 0:
            print(i)
```

Algoritmo (b) - Máximo Divisor Comum

Fluxo: Ler dois números → Testar se algum é zero → Enquanto b ≠ 0 → r = a % b → a = b, b = r → Fim.

Grafo de Fluxo:



Complexidade Ciclomática:

V(G) = 3 regiões (checagem inicial + loop com variações).

V(G) = 8 arestas - 7 nós + 2 = 3.

V(G) = 2 nós predicados + 1 = 3.

Conjunto Base de Caminhos Independentes:

 Caminho 1: Um dos números = 0 → 1 – 2 – Sim – 3 - Fim.

 Caminho 2: Números > 0, termina em poucas iterações → 1 – 2 – Não – 4 -5 – Sim – 7 – 8 – 9 – 5 – Não – 6 - Fim (ex: poucas trocas).

 Caminho 3: Números > 0, várias iterações → 1 – 2 – Não – 4 – 5 – Sim – 7 - 8 – 9 – 5 – Sim – 7 – 8 – 9 - ... - 5 – Não – 6 - Fim (múltiplas trocas).

Casos de Teste:

Caminho	Entrada	Resultado Esperado
1	**a=10, b=0**	**MDC = 10**
2	**a=48, b=18**	**MDC = 6**
3	**a=270, b=192**	**MDC = 6 (múltiplas iterações)**

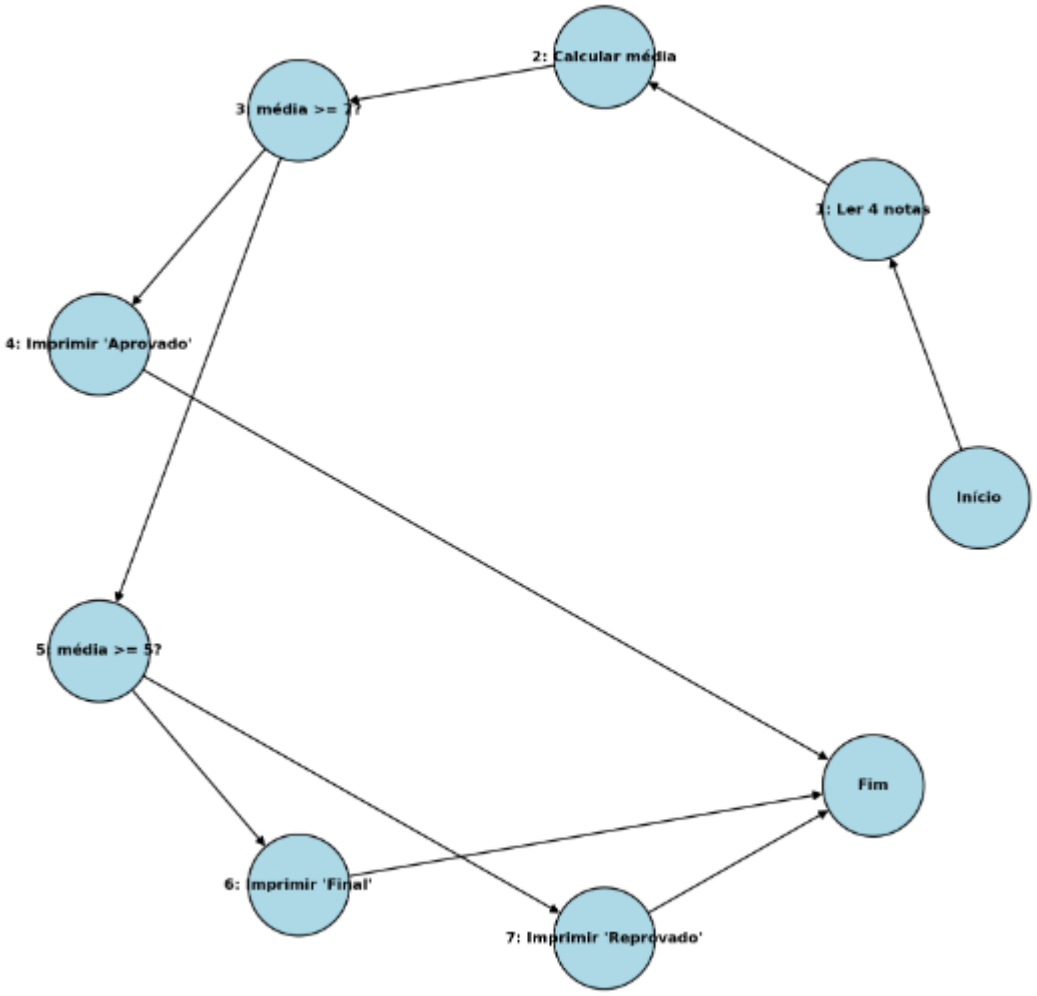
Implementação em Python:

```
def mdc(a, b):
    if b == 0:
        return a
    while b != 0:
        r = a % b
        a = b
        b = r
    return a
```

Algoritmo (c) - Notas do Aluno

Fluxo: Ler 4 notas → Calcular média → Se média >= 7 → aprovado → Senão, se média >= 5 → final → Senão → reprovado.

Grafo de Fluxo:



Complexidade Ciclomática:

V(G) = 3 regiões (decisões encadeadas).

V(G) = 6 arestas - 5 nós + 2 = 3.

V(G) = 2 nós predicados + 1 = 3.

Conjunto Base de Caminhos Independentes:

 Caminho 1: Média >= 7 → aprovado → 1 – 2 – 3 – Sim – 4 - Fim.

 Caminho 2: 5 <= média < 7 → final → 1 – 2 – 3 – Não – 5 – Sim – 6 - Fim.

 Caminho 3: Média < 5 → reprovado → 1 – 2 – 3 – Não – 5 – Não – 7 - Fim.

Casos de Teste:

Caminho	Entrada	Resultado Esperado
1	Notas = [8,7,9,10]	Aprovado
2	Notas = [6,6,5,7]	Final
3	Notas = [2,3,4,5]	Reprovado

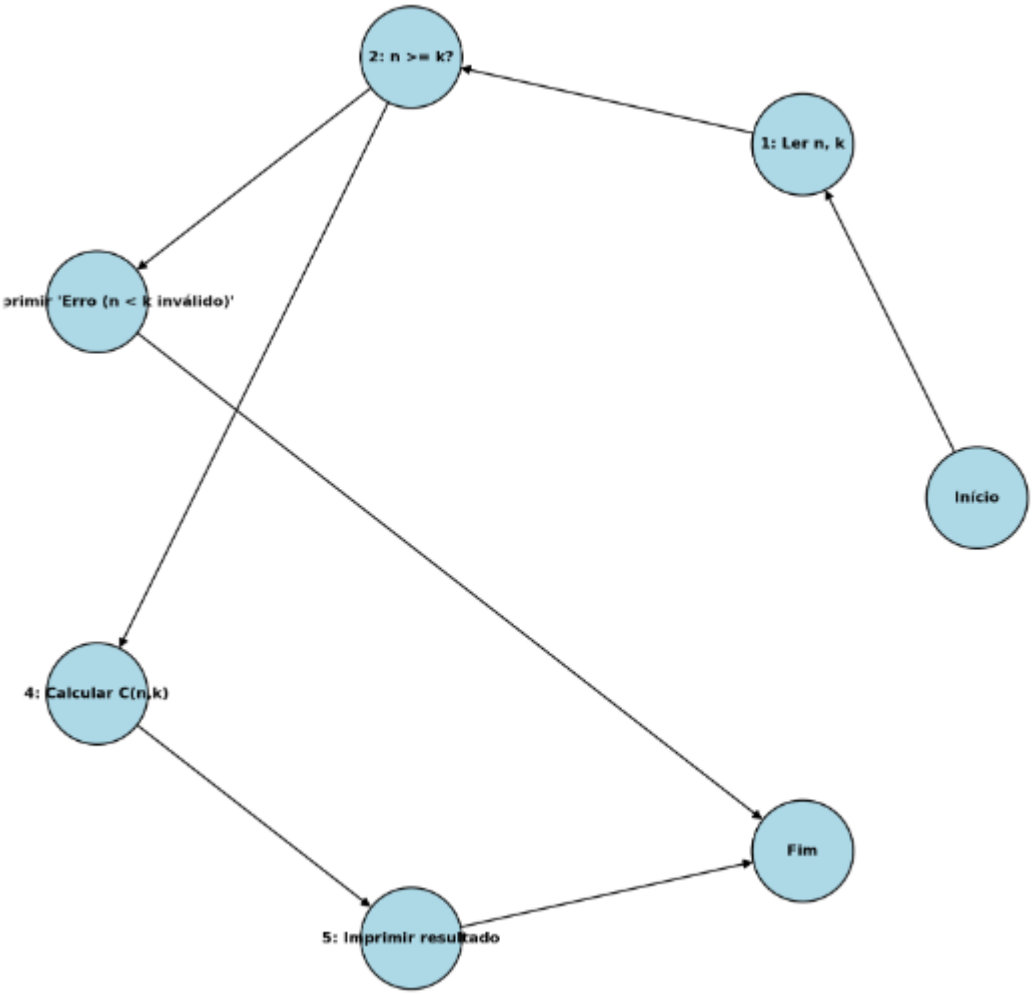
Implementação em Python:

```
def notas_aluno(notas):
    if len(notas) != 4:
        print("Erro: Deve fornecer 4 notas")
        return
    media = sum(notas) / 4
    if media >= 7:
        print("Aprovado")
    elif media >= 5:
        print("Final")
    else:
        print("Reprovado")
```

Algoritmo (d) – Combinatória

Fluxo: Ler n, k → Testar se n < k → Se sim, erro → Caso contrário, calcular C(n,k) = n! / (k!(n-k)!).

Grafo de Fluxo:



Complexidade Ciclomática:

V(G) = 2 regiões (decisão única).

V(G) = 4 arestas - 4 nós + 2 = 2.

V(G) = 1 nó predicado + 1 = 2.

Conjunto Base de Caminhos Independentes:

 Caminho 1: Entrada inválida (n < k) → 1 – 2 – Não – 3 - Fim.

 Caminho 2: Entrada válida (n >= k) → calcular combinatória → 1 – 2 – Sim -4 – 5 - Fim.

Casos de Teste:

Caminho	Entrada	Resultado Esperado
1	**n=3, k=5**	**Erro (n < k inválido)**
2	**n=5, k=2**	**C(5,2) = 10**

Implementação em Python:

```
import math
def combinatoria(n, k):
    if n < k:
        print("Erro (n < k inválido)")
        return
    print(math.comb(n, k))
```

valores e totais apropriados.

PLANOS DE TESTE A SER DESCRITO:

ITENS A TESTAR / ABORDAGEM:

N°	Item	Especificação
	Algoritmo de Divisores de um Número	Ler um número inteiro e listar todos os seus divisores
2	Algoritmo de Máximo Divisor Comum (MDC)	Ler dois números e calcular o MDC pelo método de Euclides
3	Algoritmo de Notas do Aluno	Ler 4 notas e classificar como aprovado, final ou reprovado
4	Algoritmo de Combinatória	Calcular C(n,k) com n ≥ k, ou retornar erro se n < k

ABORDAGEM: 1- Teste estrutural com base no grafo de fluxo e caminhos independentes, cobrindo entradas inválidas, número 1 e número com múltiplos divisores

2- Teste estrutural cobrindo casos com zero, poucas iterações e múltiplas iterações

3- Teste estrutural cobrindo as três condições de decisão

4- Teste estrutural cobrindo entrada inválida e válida

CRONOGRAMA DE TESTES

ID	Tarefa	Início		Fim	Esforço	Pré	Pessoa	Obs
01	Preparar ambiente de teste	18/09/2025		18/09/2025	30min	null	Lucas	null
02	Executar testes do Algoritmo de Divisores	18/09/2025		18/09/2025	30min	01	Lucas	null
03	Executar testes do Algoritmo de MDC	18/09/2025		18/09/2025	30min	01	Gabriel	null
04	Executar testes do Algoritmo de Notas	18/09/2025		18/09/2025	30min	01	Thiago	null
05	Executar testes do Algoritmo de Combinatória	18/09/2025		18/09/2025	30min	01	Lucas	null

AMBIENTE DE TESTE

Ambiente	Descrição
Hardware	AMD Ryzen 7 5700X3D 8-Core Processor
Software	Sistema Operacional Windows 10, Python 3.10
Ferramental	Editor de código (VS Code)

IDENTIFICAÇÃO DE CASO DE TESTE / IDENTIFICAÇÃO DE PROCEDIMENTO DE TESTE

N°	Caso de Teste	Identificação do Caso de Teste		Procedimento	Identificação do Procedimento de Teste
1	Divisores de um Número	CT-DIV		Executar o programa com entradas -5, 1 e 6 e verificar se a saída corresponde aos divisores esperados ou mensagem de erro	PT-DIV
2	Máximo Divisor Comum	CT-MDC		Executar o programa com pares (10,0), (48,18) e (270,192) e validar o resultado do MDC	PT-DIV
3	Notas do Aluno	CT-NOT		Executar o programa com conjuntos de notas [8,7,9,10], [6,6,5,7] e [2,3,4,5] e validar classificação	PT-NOT
4	Combinatória	CT-COMB		Executar o programa com entradas (3,5) e (5,2) e validar se retorna erro ou valor correto	PT-COMB
5	Validação Geral	CT-GERAL		Executar todos os algoritmos em sequência para validar integração e consistência	PT-GERAL

CASO DE TESTE:

Identificação	CT-MDC	
Itens a Testar	Algoritmo de MDC	
Entradas	**Campo**	**Valor**
	10,0	Inteiro
	48,18	Inteiro
	270,192	Inteiro
Saídas Esperadas	**Campo**	**Valor**
	Inteiro	10
	Inteiro e Inteiro	6 e 6

Ambiente	Conforme ambiente
Procedimento	Executar com a=10,b=0 / Executar com a=48,b=18 / Executar com a=48,b=18
Dependência	Nenhuma

Identificação	CT-DIV	
Itens a Testar	Algoritmo de Divisores	
Entradas	**Campo**	**Valor**
	-5	Inteiro
	1	Inteiro
	6	Inteiro
Saídas Esperadas	**Campo**	**Valor**
	Texto	"Entrada inválida"
	Lista	[1] e [1,2,3,6]
Ambiente	Conforme ambiente	
Procedimento	Executar com n=-5 / Executar com n=1 / Executar com n=6	
Dependência	Nenhuma	

Identificação	CT-NOT	
Itens a Testar	Algoritmo de Notas	
Entradas	**Campo**	**Valor**
	[8,7,9,10]	Lista
	[6,6,5,7]	Lista
	[2,3,4,5]	Lista
Saídas Esperadas	**Campo**	**Valor**
	Texto	"Aprovado"

	Texto e Texto	"Final" e "Reprovado"
Ambiente	Conforme ambiente	
Procedimento	Executar com notas	
Dependência	Nenhuma	

Identificação	CT-COMB	
Itens a Testar	Algoritmo de Combinatória	
Entradas	**Campo**	**Valor**
	3,5	Inteiro
	5,2	Inteiro
Saídas Esperadas	**Campo**	**Valor**
	Texto	"Entrada inválida"
	Inteiro	10
Ambiente	Conforme ambiente	
Procedimento	Executar com n=3,k=5 / Executar com n=5,k=2	
Dependência	Nenhuma	

PROCEDIMENTO DE TESTE:

Identificação	PT-DIV
Objetivo	Validar se o algoritmo retorna corretamente os divisores de um número ou mensagem de erro para entrada inválida
Requisitos	Código implementado, ambiente configurado
Fluxo	1. Executar com n=-5 → verificar mensagem de erro. 2. Executar com n=1 → verificar lista [1]. 3. Executar com n=6 → verificar lista [1,2,3,6]

Identificação	PT-MDC

Objetivo	Validar se o algoritmo calcula corretamente o MDC de dois números
Requisitos	Código implementado, ambiente configurado
Fluxo	1. Executar com (10,0) → verificar 10. 2. Executar com (48,18) → verificar 6. 3. Executar com (270,192) → verificar 6

Identificação	PT-NOT
Objetivo	Validar se o algoritmo classifica corretamente o aluno
Requisitos	Código implementado, ambiente configurado
Fluxo	1. Executar com [8,7,9,10] → verificar "Aprovado". 2. Executar com [6,6,5,7] → verificar "Final". 3. Executar com [2,3,4,5] → verificar "Reprovado"

Identificação	PT-COMB
Objetivo	Validar se o algoritmo calcula corretamente a combinatória ou retorna erro para entrada inválida
Requisitos	Código implementado, ambiente configurado
Fluxo	1. Executar com (3,5) → verificar mensagem de erro. 2. Executar com (5,2) → verificar 10

Identificação	PT-GERAL
Objetivo	Validar a execução sequencial de todos os algoritmos
Requisitos	Código implementado, ambiente configurado
Fluxo	Executar todos os casos de teste anteriores em sequência e verificar consistência