# 1 Introdução

Este relatório descreve a implementação do problema clássico da Torre de Hanói utilizando estruturas de dados do tipo pilha em C++. O objetivo foi desenvolver uma solução que não apenas resolvesse o problema, mas também apresentasse visualmente o estado das torres durante a execução.

# 2 Estruturas de Dados Utilizadas

## 2.1 Classe `pilha`

- **Propósito**: Implementação de uma pilha estática com operações básicas
- **Atributos**:
  - `elementos`: Array dinâmico para armazenar os valores
  - `tamanho_lista`: Capacidade máxima da pilha
  - `topo`: Índice do elemento no topo (-1 para pilha vazia)
- **Métodos Principais**:
  - `empilha()`: Adiciona um elemento no topo (O(1))
  - `desempilha()`: Remove e retorna o elemento do topo (O(1))
  - `ler_topo()`: Retorna o elemento do topo sem removê-lo (O(1))
  - `troca()`: Troca os dois elementos superiores (O(1))
- **Complexidade**:
  - Espaço: O(n) para armazenar os elementos
  - Tempo: Todas as operações principais são O(1)

# 3 Divisão de Módulos

O código está organizado em três componentes principais:

1. **Classe `pilha`**: Implementa todas as operações básicas de uma pilha
2. **Funções de Visualização**:
   - `repetir_string()`: Auxiliar para formatação visual
   - `imprimir_pilhas()`: Exibe o estado atual das três torres
3. **Lógica da Torre de Hanói**:
   - `mover_disco()`: Move um disco entre torres
   - `resolver_torre_hanoi()`: Implementa o algoritmo recursivo clássico

# 4 Descrição das Rotinas e Funções

## 4.1 `imprimir_pilhas()`

- **Propósito**: Exibir graficamente o estado das três torres
- **Funcionamento**:
  - Utiliza cópias temporárias das pilhas para não alterar o estado original
  - Imprime cada disco como uma linha de caracteres #, centralizado no pino
- **Complexidade**: O($n^2$) devido à necessidade de processar cada disco em cada torre

## 4.2 `resolver_torre_hanoi()`

- **Algoritmo**: Implementação recursiva clássica:
  1. Move n-1 discos da origem para a auxiliar
  2. Move o disco restante para o destino
  3. Move n-1 discos da auxiliar para o destino
- **Parâmetros Adicionais**:
  - `origem2`, `destino2`, `auxiliar2`: Referências constantes às pilhas originais para visualização
  - `intervalo_impressao`: Controla a frequência de exibição
- **Complexidade**:
  - Tempo: O(2) (típico do problema de Hanói)
  - Espaço: O(n) devido à pilha de chamadas recursivas

# 5 Problemas e Observações

- **Eficiência de Visualização**:
  - A criação de pilhas temporárias em `imprimir_pilhas()` consome recursos desnecessários
  - Solução possível: Implementar um iterador para percorrer a pilha sem modificá-la
- **Controle de Impressão**:
  - O parâmetro `intervalo_impressao` não funciona como esperado devido ao contador estático

  - Melhoria: Usar um contador passado como parâmetro recursivo

- **Limitações da Pilha**:
  - A implementação atual não permite redimensionamento
  - Em cenários reais, seria interessante implementar uma versão dinâmica

- **Experiência do Usuário**:
  - O `cin.ignore()` força uma pausa a cada passo, o que pode ser inconveniente
  - Sugestão: Adicionar controle manual (tecla pressionada) ou intervalo de tempo

# 6 Conclusão

O projeto implementou com sucesso a Torre de Hanói utilizando pilhas estáticas, demonstrando:

- A aplicação prática de estruturas de dados básicas
- A eficácia da recursão para problemas de divisão e conquista
- Desafios no balanceamento entre funcionalidade e desempenho visual

**Principais conquistas**:

- Visualização clara do estado das torres em cada passo
- Implementação correta do algoritmo clássico