Relatório: Solução da Torre de Hanói com Análise de Tempo de Execução

Aluno: Paulo Willian Costa Rodrigues

Curso: Sistemas de Informação

Disciplina: Algoritmos e Complexidade

Professor: Alexandre Freitas

---

1. Introdução

O presente trabalho tem como objetivo implementar uma solução para o problema clássico da Torre de Hanói utilizando a linguagem de programação Python. A implementação armazena os movimentos em listas e realiza a medição do tempo de execução para diferentes execuções do algoritmo, possibilitando uma análise de desempenho do sistema.

---

2. Objetivos

Implementar o algoritmo da Torre de Hanói utilizando listas para guardar os movimentos.

Realizar no mínimo 10 execuções do algoritmo para analisar o tempo de execução.

Apresentar um relatório com os resultados obtidos.

---

3. Desenvolvimento

3.1. Algoritmo Utilizado

O algoritmo utilizado segue a estratégia recursiva clássica para resolução da Torre de Hanói. Ele move nos discos da torre de origem para a torre de destino, utilizando uma torre auxiliar. Cada movimento é registrado em uma lista.

3.2. Estrutura do Código

O código está dividido em três funções principais:

hanoi(...): função recursiva que resolve o problema e armazena os movimentos em uma lista.

executar_testes(...): executa o algoritmo 10 vezes, medindo o tempo com a biblioteca time.

gerar_relatorio(...): exibe os tempos de execução, além de calcular média, mínimo e máximo.

O teste foi feito com 10 discos, o que representa uma complexidade considerável para observar variações de tempo.

---

4. Resultados

Abaixo estão os tempos de execução obtidos nas 10 execuções com 10 discos:

```
PS C:\Users\paulo\Music\WORKSPACE> & C:/Users/paulo/AppData/Local/Programs/Python/Python313/python.exe c:/Users/paulo/Music/WORKSPACE/TRABALHOFACULDADE/Torredehanoi.py
Execução 1: 0.000162 segundos | Movimentos: 1023
Execução 2: 0.000314 segundos | Movimentos: 1023
Execução 3: 0.000086 segundos | Movimentos: 1023
Execução 4: 0.000075 segundos | Movimentos: 1023
Execução 5: 0.000079 segundos | Movimentos: 1023
Execução 6: 0.000073 segundos | Movimentos: 1023
Execução 7: 0.000185 segundos | Movimentos: 1023
Execução 8: 0.000081 segundos | Movimentos: 1023
Execução 9: 0.000073 segundos | Movimentos: 1023
Execução 10: 0.000116 segundos | Movimentos: 1023

Relatório de Execução
--------------------
Execução 1: 0.000162 segundos
Execução 2: 0.000314 segundos
Execução 3: 0.000086 segundos
Execução 4: 0.000075 segundos
Execução 5: 0.000079 segundos
Execução 6: 0.000073 segundos
Execução 7: 0.000185 segundos
Execução 8: 0.000081 segundos
Execução 9: 0.000073 segundos
Execução 10: 0.000116 segundos

Tempo médio: 0.000124 segundos
Tempo máximo: 0.000314 segundos
Tempo mínimo: 0.000073 segundos
PS C:\Users\paulo\Music\WORKSPACE>
```

---

5. Conclusão

A implementação da Torre de Hanói com listas mostrou-se eficiente e clara. Mesmo com o aumento do número de discos, o algoritmo manteve uma boa performance nas execuções. A utilização de listas para armazenar os movimentos possibilitou uma verificação direta do número de passos realizados pelo sistema.

A medição de tempos permitiu observar pequenas variações de desempenho, principalmente influenciadas por recursos do sistema operacional em tempo de execução.

---

6. Código Fonte

```python
import time

def hanoi(n, origem, destino, auxiliar, movimentos):
    if n == 1:
        movimentos.append((origem, destino))
    else:
        hanoi(n - 1, origem, auxiliar, destino, movimentos)
        movimentos.append((origem, destino))
        hanoi(n - 1, auxiliar, destino, origem, movimentos)

def executar_testes(numero_discos, repeticoes=10):
    tempos_execucao = []

    for i in range(repeticoes):
        movimentos = []
        inicio = time.time()
        hanoi(numero_discos, 'A', 'C', 'B', movimentos)
        fim = time.time()
        duracao = fim - inicio
        tempos_execucao.append(duracao)
        print(f"Execução {i+1}: {duracao:.6f} segundos | Movimentos: {len(movimentos)}")

    return tempos_execucao

def gerar_relatorio(tempos_execucao):
    media = sum(tempos_execucao) / len(tempos_execucao)
    maximo = max(tempos_execucao)
    minimo = min(tempos_execucao)

    print("\nRelatório de Execução")
    print("---------------------")
    for i, tempo in enumerate(tempos_execucao, 1):
        print(f"Execução {i}: {tempo:.6f} segundos")

    print(f"\nTempo médio: {media:.6f} segundos")
    print(f"Tempo máximo: {maximo:.6f} segundos")
    print(f"Tempo mínimo: {minimo:.6f} segundos")


numero_discos = 10
tempos = executar_testes(numero_discos)
gerar_relatorio(tempos)
```

---

7. Explicação do Código

7.1 Importação da biblioteca de tempo

```python
import time
```

- Importa o módulo time, que permite medir o tempo de execução do algoritmo com precisão.

7.2. Função principal da Torre de Hanói

```python
def hanoi(n, origem, destino, auxiliar, movimentos):
```

- Define a função recursiva hanoi, com:
- n: número de discos.
- origem: torre de onde os discos devem sair.
- destino: torre para onde os discos devem ir.
- auxiliar: torre auxiliar usada para movimentar os discos.
- movimentos: lista onde serão armazenados os movimentos realizados.

```
    if n == 1:
        movimentos.append((origem, destino))
```

- Caso base da recursão: se houver apenas 1 disco, move diretamente da torre de origem para a torre de destino e registra esse movimento na lista movimentos.

```
else:
    hanoi(n - 1, origem, auxiliar, destino, movimentos)
    movimentos.append((origem, destino))
    hanoi(n - 1, auxiliar, destino, origem, movimentos)
```

- Caso recursivo:

1- Move n-1 discos da origem para a auxiliar.
2- Move o maior disco da origem para o destino.
3- Move os n-1 discos da auxiliar para o destino.

7.3 Função para executar os testes

```
def executar_testes(numero_discos, repeticoes=10):
```

- Função que executa o algoritmo várias vezes para medir o tempo de execução.
- Recebe:
- numero_discos: quantidade de discos.
- repeticoes: quantas vezes o teste será executado (padrão é 10).

```
 tempos_execucao = []
```

- Cria uma lista para armazenar os tempos de cada execução.

```
 for i in range(repeticoes):
```

- Faz um laço para repetir o teste 10 vezes.

```
movimentos = []
inicio = time.time()
hanoi(numero_discos, 'A', 'C', 'B', movimentos)
fim = time.time()
```

- movimentos: lista que vai guardar os movimentos dessa execução.
- inicio e fim: capturam o tempo antes e depois de rodar o algoritmo.
- 'A', 'C' e 'B' são os nomes das torres (origem, destino e auxiliar).

```
duracao = fim - inicio
tempos_execucao.append(duracao)
print(f"Execução {i+1}: {duracao:.6f} segundos | Movimentos: {len(movimentos)}")
```

- Calcula o tempo de execução (duracao) e adiciona na lista tempos_execucao.
- Mostra na tela o número da execução, tempo e quantos movimentos foram feitos.

```
return tempos_execucao
```

- Retorna a lista com todos os tempos das execuções.

7.4 Função para gerar o relatório dos tempos

```
def gerar_relatorio(tempos_execucao):
```

- Função que recebe a lista de tempos e imprime um relatório com média, máximo e mínimo.

```
media = sum(tempos_execucao) / len(tempos_execucao)
maximo = max(tempos_execucao)
minimo = min(tempos_execucao)
```

- Calcula:
- media: tempo médio.
- maximo: maior tempo registrado.
- minimo: menor tempo registrado.

```
print("\nRelatório de Execução")
print("----------------------")
```

- Mostra um título para o relatório.

```
for i, tempo in enumerate(tempos_execucao, 1):
    print(f"Execução {i}: {tempo:.6f} segundos")
```

- Mostra cada tempo de execução formatado com 6 casas decimais.

```
print(f"\nTempo médio: {media:.6f} segundos")
print(f"Tempo máximo: {maximo:.6f} segundos")
print(f"Tempo mínimo: {minimo:.6f} segundos")
```

- Exibe os dados finais: tempo médio, máximo e mínimo.

7.5 Chamada do programa principal

```
numero_discos = 10
tempos = executar_testes(numero_discos)
gerar_relatorio(tempos)
```

- Define o número de discos.
- Executa os testes e coleta os tempos.
- Gera o relatório com os resultados.