**Resolvendo a recorrência Torre de Hanoi**

Temos .

**Método de expansão** (**unfolding**):

...

Escolha e use:

**Logo, a forma fechada é**

**Prova por indução (verificação)**

Base : ✔

**Hipótese**: suponha para algum .

Passo: para ,

Logo vale para . Pelo princípio da indução, vale para todo .

**Prova de que é mínimo (argumento)**

Qualquer solução que mova discos precisa, em algum momento, mover o maior disco do pino fonte para o destino. Antes desse movimento, os discos acima dele devem estar em outro pino (não no destino). Assim já são necessárias pelo menos movimentações antes e depois — total . Com a recorrência saturada, a solução recursiva atinge esse valor, logo é mínimo.

**Complexidade**

Número de movimentos — cresce exponencialmente. Complexidade de tempo (algoritmo que gera os movimentos) é .

**Solução em Python**

def **hanoi**(n, source, target, aux):

if n == 0:

return

**hanoi**(n-1, source, aux, target)

print(f"Move disk from {source} to {target}")

**hanoi**(n-1, aux, target, source)

# Exemplo: **hanoi**(3, 'A', 'C', 'B')

Chamando **hanoi**(3, 'A', 'C', 'B') **#** **produz 7 movimentos (pois 23 −1 = 7).**

Resolvendo pelo **método da árvore de recursão** para a recorrência

**Intuição com árvore**

Cada chamada T(n) gera duas chamadas T(n-1) (dois filhos) e um custo **1** (o movimento do maior disco). Representamos isso como uma árvore onde cada nó corresponde a uma chamada e carrega custo (exceto as folhas onde — elas não acrescentam custo).

**Estrutura por níveis**

* Nível 0 (raiz): nó com custo total .
* Nível 1: nós, cada um com custo → custo total .
* Nível 2: nós → custo total .
* ...
* Nível : nós → custo total .

A árvore continua até que as chamadas atinjam . Para partir de , há exatamente níveis com custo (níveis a ); em nível estão as chamadas que custam .

Portanto o custo total é a soma dos custos por nível:

Essa soma é uma progressão geométrica:

**Exemplo explícito (n = 3)**

Árvore (cada nó mostra custo 1):

[1] T(3)

/ \

[1]T(2) [1]T(2)

/ \ / \

[1]T(1)[1]T(1) [1]T(1)[1]T(1)

/ \ / \ / \ / \

T(0)T(0) T(0)T(0) ... (**custos 0 nas folhas**)

Custos por nível: . Soma = .

**Conclusão**

A árvore de recursão mostra claramente que o custo por nível dobra, com níveis não-nulos, somando . Logo a solução é

com complexidade assintótica .