Matemática Concreta

Problemas Recorrentes

Dr. A. Riker
Universidade Federal do Pará (UFPA)
afr@ufpa.br

2021.PL03

Problemas Recorrentes

Problemas Recorrentes

Torre de Hanoi

- Torre de Hanói é um quebra-cabeça matemático em que temos três hastes e n discos. O objetivo do quebra-cabeça é mover toda a pilha para outra haste, obedecendo às seguintes regras simples:
- Apenas um disco pode ser movido por vez.
- Cada movimento consiste em retirar o disco superior de uma das pilhas e colocá-lo no topo de outra pilha, ou seja, um disco só pode ser movido se for o disco mais alto de uma pilha.
- Nenhum disco pode ser colocado em cima de um disco menor.

Torre de Hanoi

The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883.

Using mathematical induction one can prove that

For the Tower of Hanoi puzzle with $n \geqslant 0$ (and 3 pegs), the minimum number of moves needed

$$T_n = 2^n - 1.$$

Let's look at the example borrowed from Martin Hofmann and Berteun Damman.

Torre de Hanoi

The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883.

Using mathematical induction one can prove that

For the Tower of Hanoi puzzle with $n \geqslant 0$ (and 3 pegs), the minimum number of moves needed

$$T_n = 2^n - 1.$$

Let's look at the example borrowed from Martin Hofmann and Berteun Damman.

Torre de Hanoi

The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883.

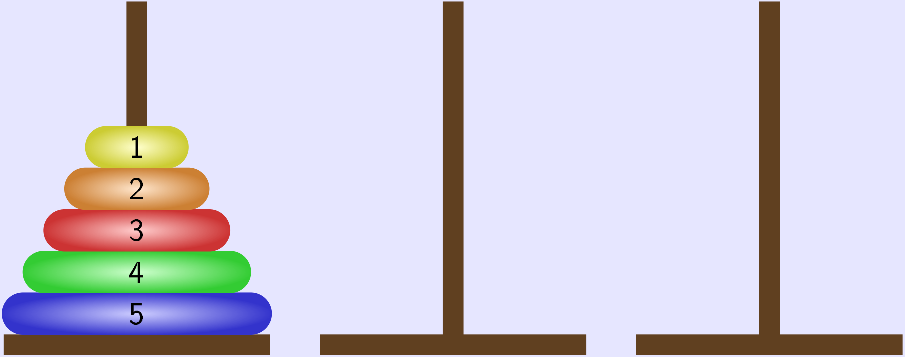
Using mathematical induction one can prove that

For the Tower of Hanoi puzzle with $n \geqslant 0$ (and 3 pegs), the minimum number of moves needed

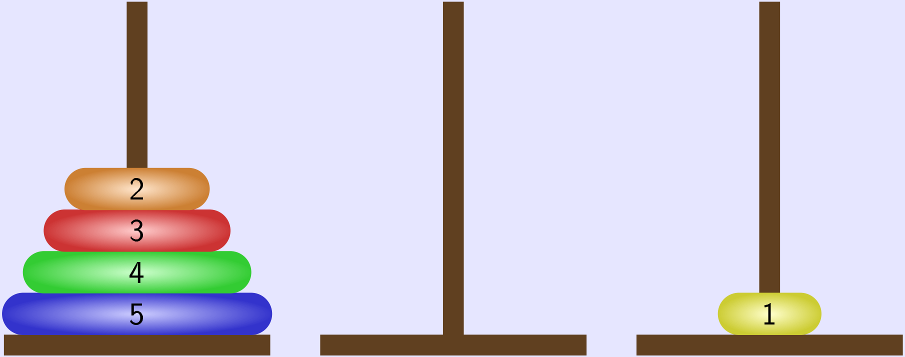
$$T_n = 2^n - 1.$$

Let's look at the example borrowed from Martin Hofmann and Berteun Damman.

Torre de Hanoi

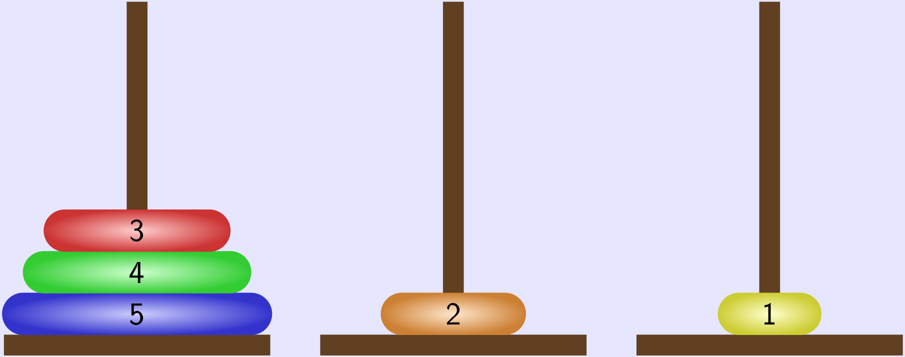


Torre de Hanoi



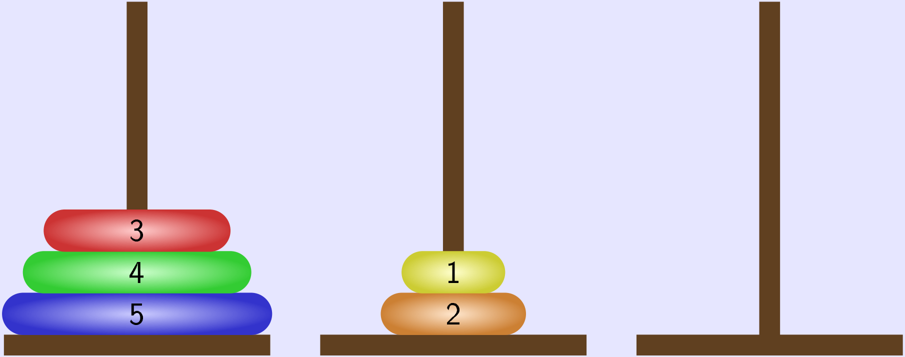
Moved disc from pole 1 to pole 3.

Torre de Hanoi



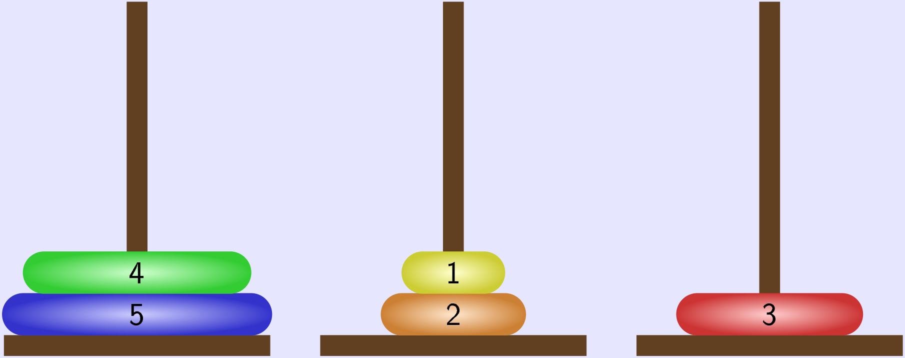
Moved disc from pole 1 to pole 2.

Torre de Hanoi



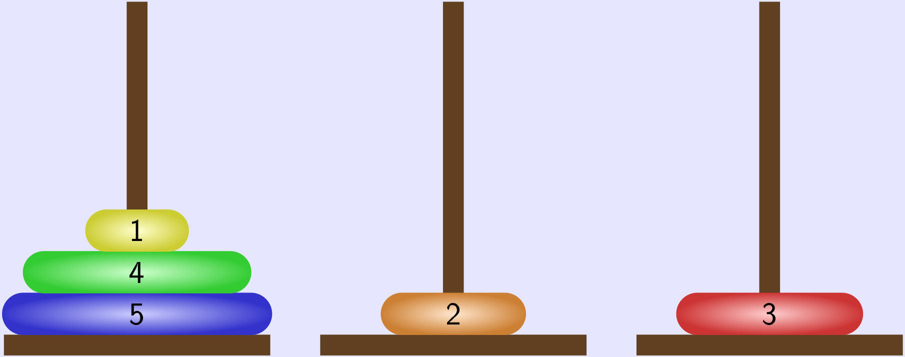
Moved disc from pole 3 to pole 2.

Torre de Hanoi



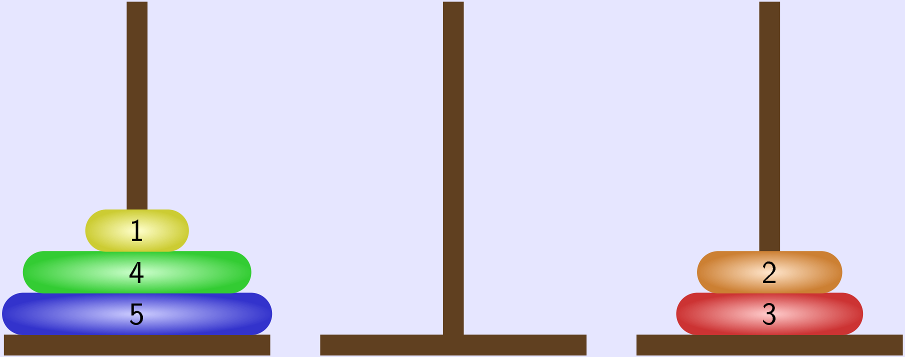
Moved disc from pole 1 to pole 3.

Torre de Hanoi



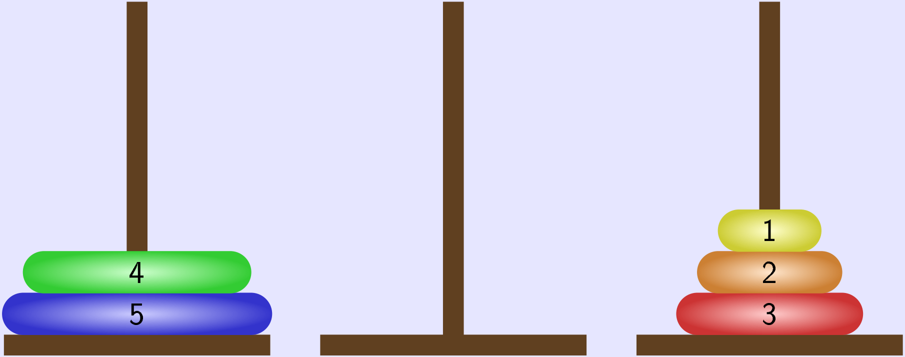
Moved disc from pole 2 to pole 1.

Torre de Hanoi



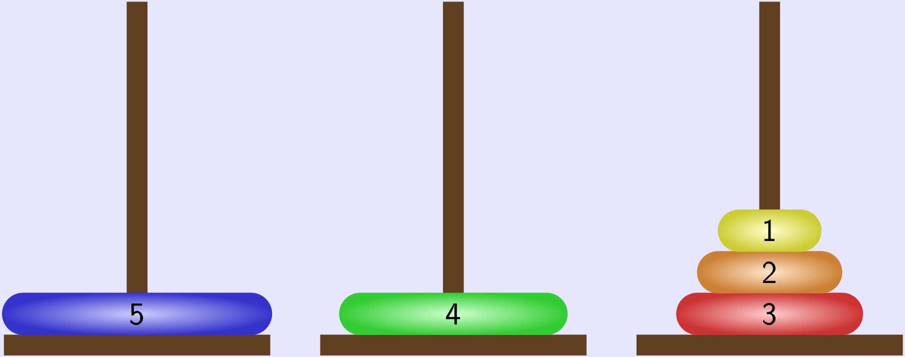
Moved disc from pole 2 to pole 3.

Torre de Hanoi



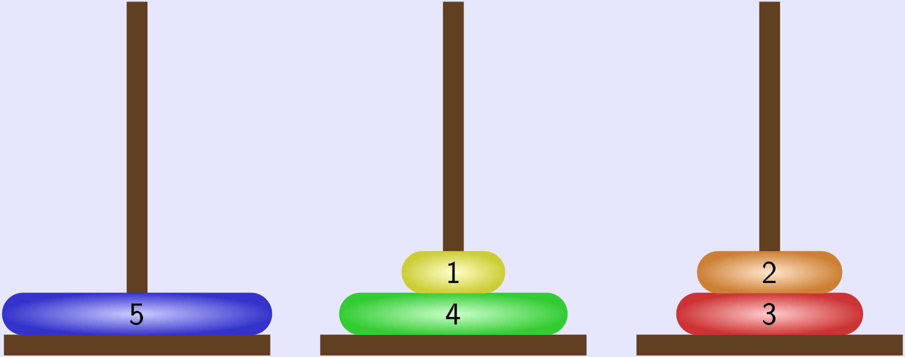
Moved disc from pole 1 to pole 3.

Torre de Hanoi



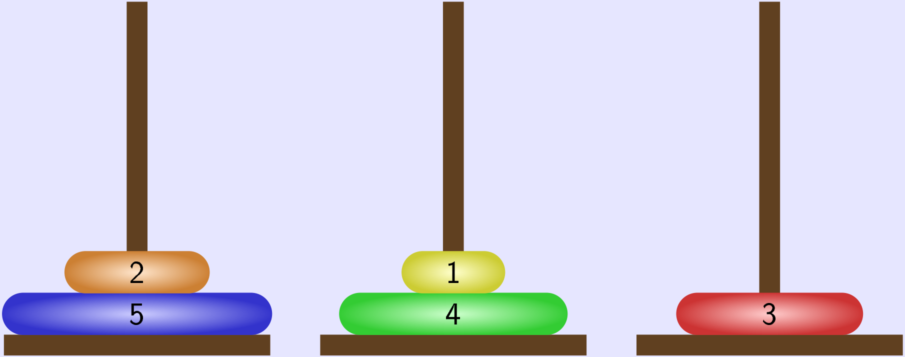
Moved disc from pole 1 to pole 2.

Torre de Hanoi



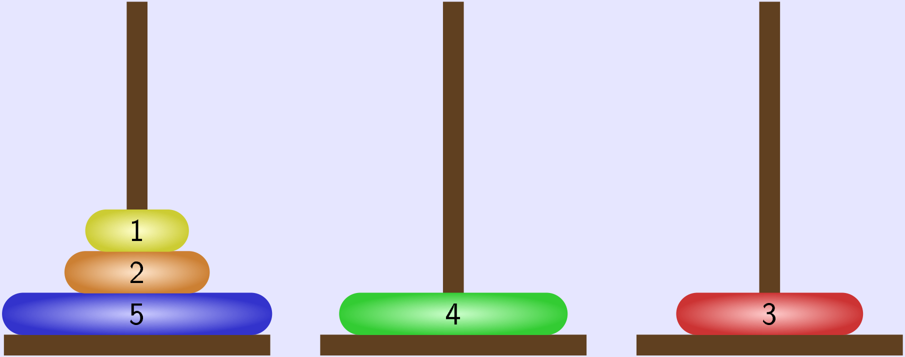
Moved disc from pole 3 to pole 2.

Torre de Hanoi



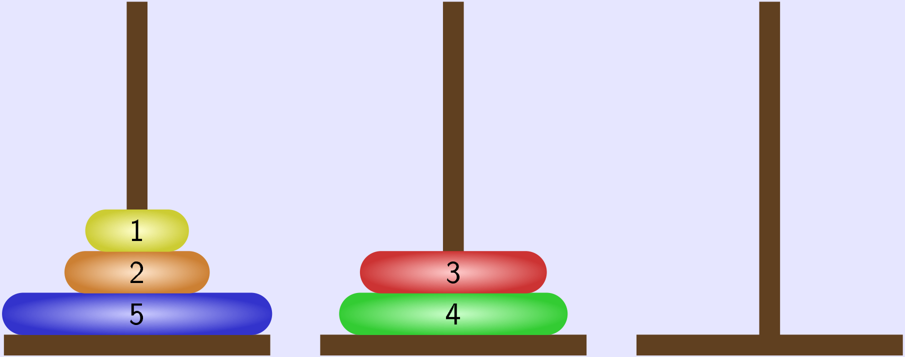
Moved disc from pole 3 to pole 1.

Torre de Hanoi



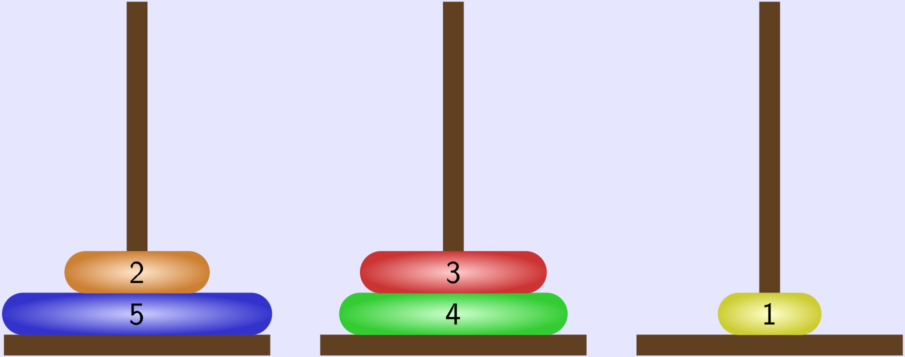
Moved disc from pole 2 to pole 1.

Torre de Hanoi



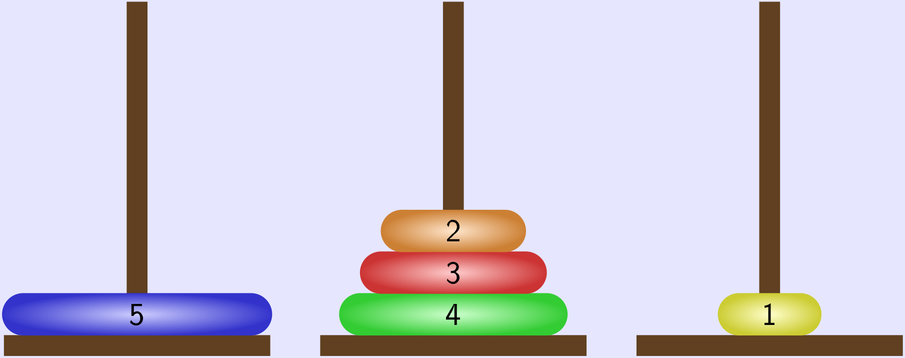
Moved disc from pole 3 to pole 2.

Torre de Hanoi



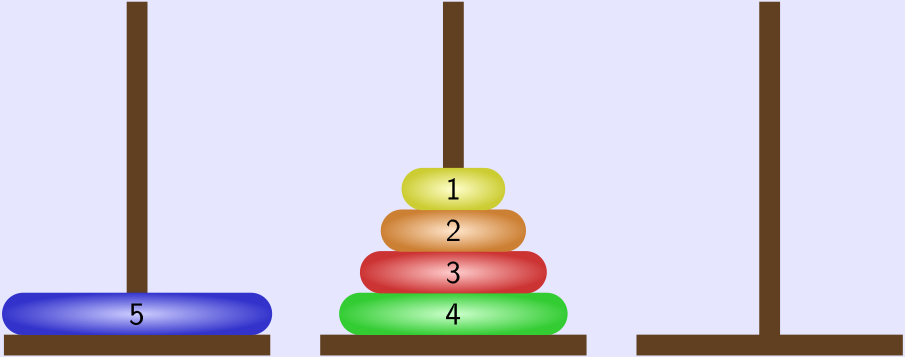
Moved disc from pole 1 to pole 3.

Torre de Hanoi



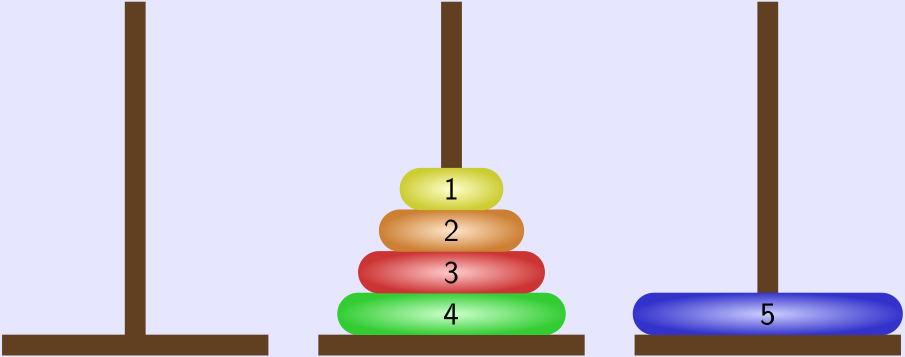
Moved disc from pole 1 to pole 2.

Torre de Hanoi



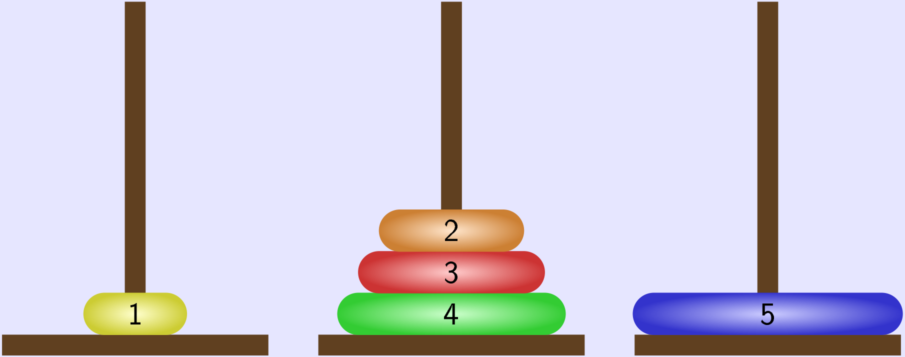
Moved disc from pole 3 to pole 2.

Torre de Hanoi



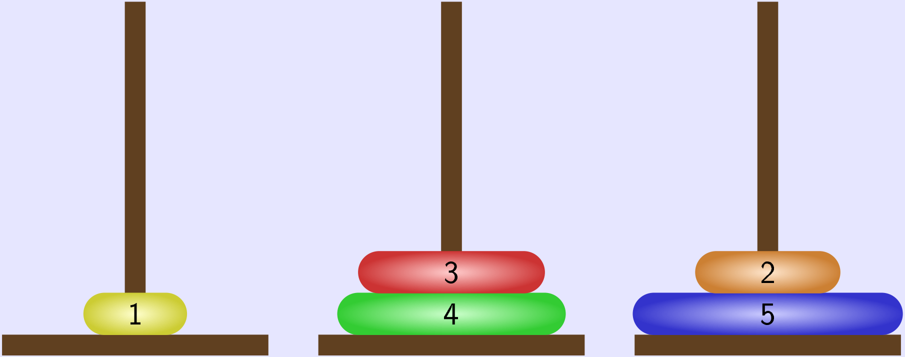
Moved disc from pole 1 to pole 3.

Torre de Hanoi



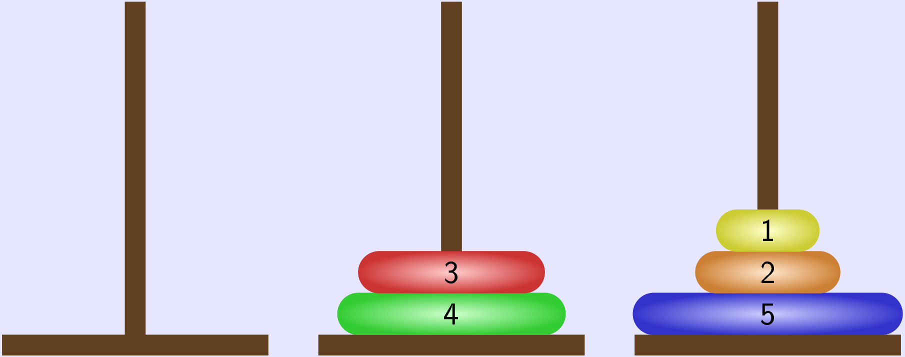
Moved disc from pole 2 to pole 1.

Torre de Hanoi



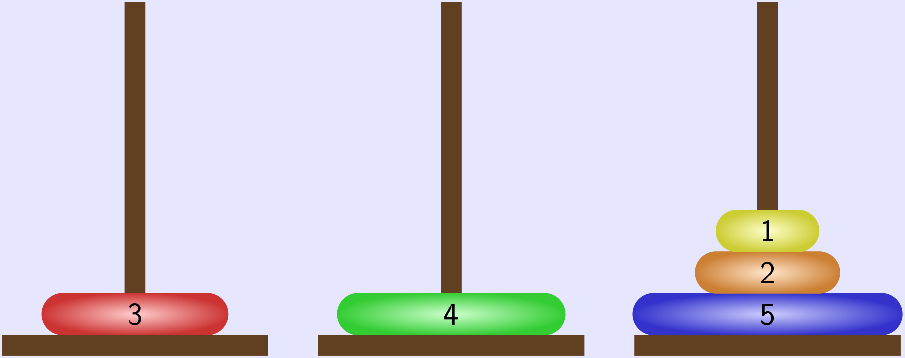
Moved disc from pole 2 to pole 3.

Torre de Hanoi



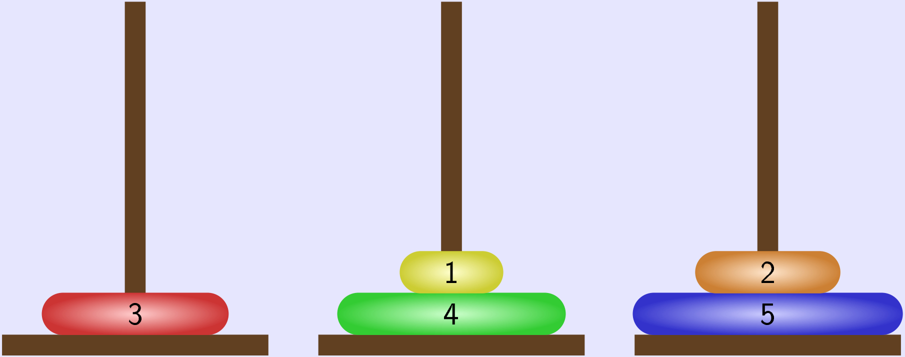
Moved disc from pole 1 to pole 3.

Torre de Hanoi



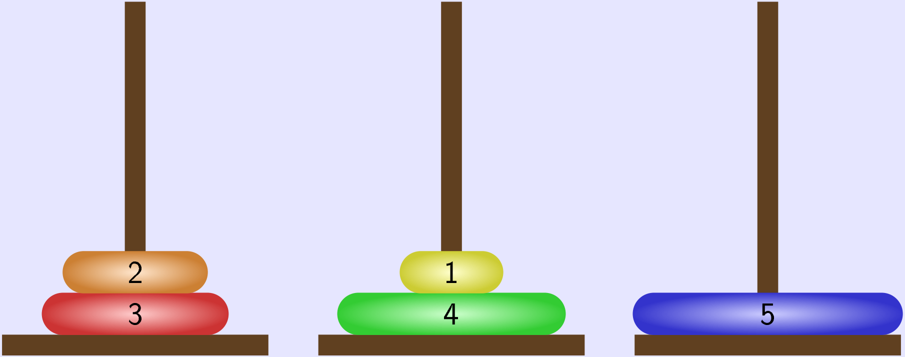
Moved disc from pole 2 to pole 1.

Torre de Hanoi



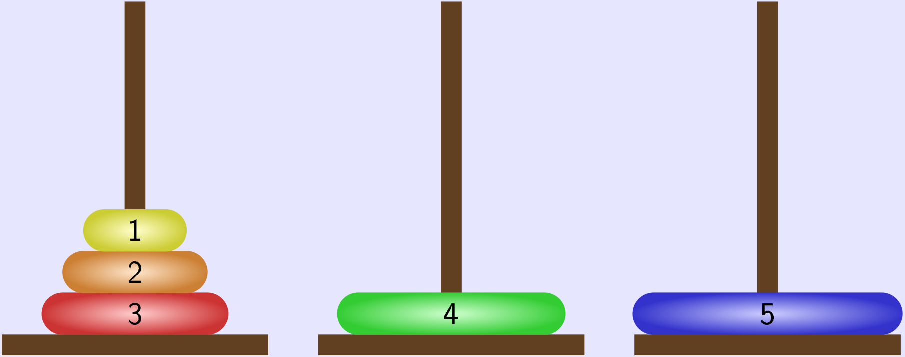
Moved disc from pole 3 to pole 2.

Torre de Hanoi



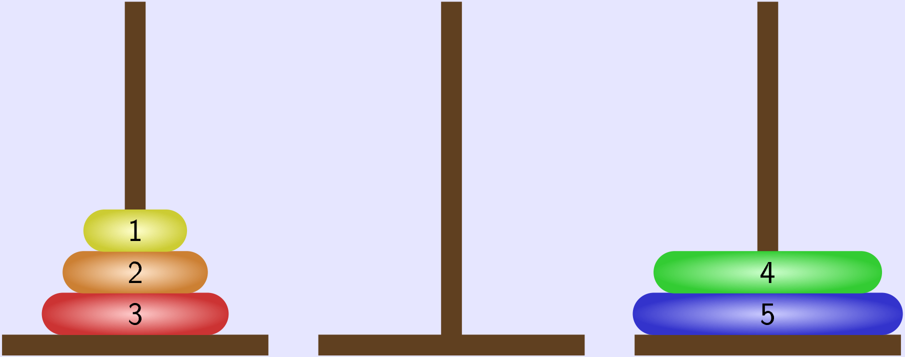
Moved disc from pole 3 to pole 1.

Torre de Hanoi



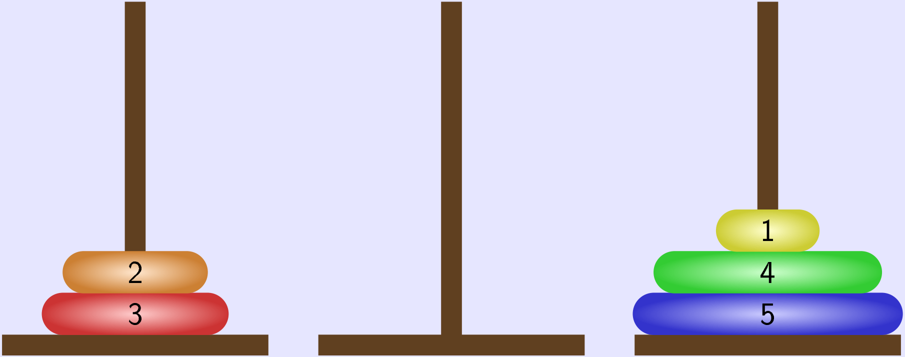
Moved disc from pole 2 to pole 1.

Torre de Hanoi



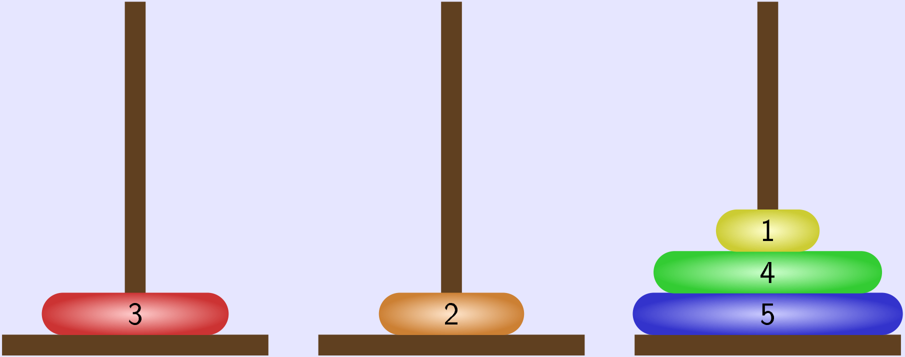
Moved disc from pole 2 to pole 3.

Torre de Hanoi



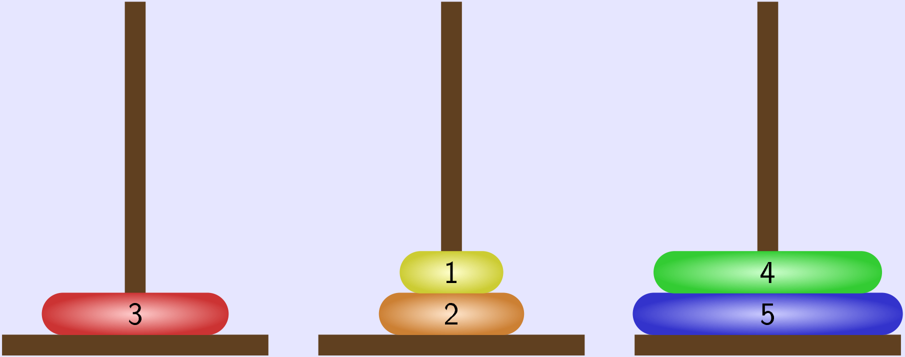
Moved disc from pole 1 to pole 3.

Torre de Hanoi



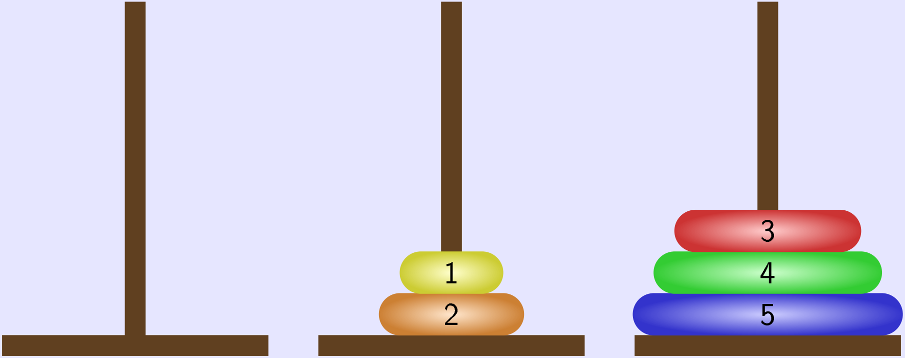
Moved disc from pole 1 to pole 2.

Torre de Hanoi



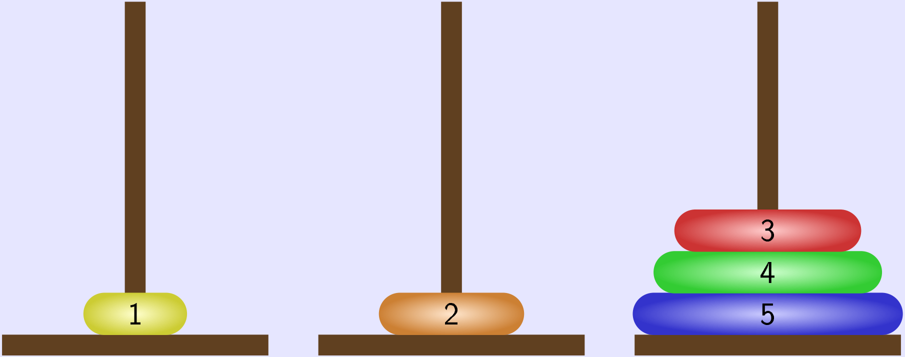
Moved disc from pole 3 to pole 2.

Torre de Hanoi



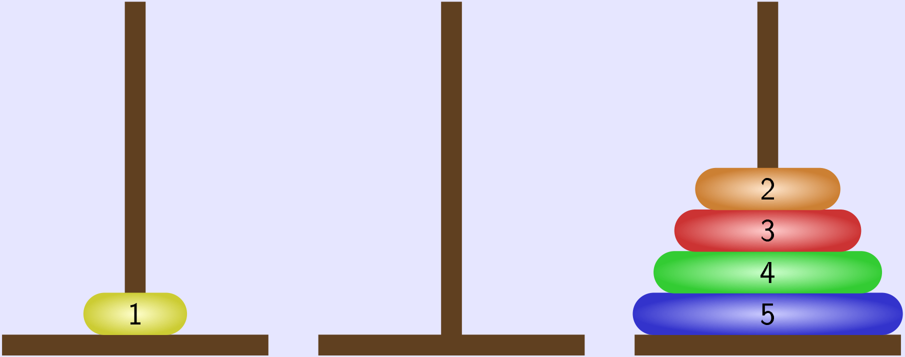
Moved disc from pole 1 to pole 3.

Torre de Hanoi



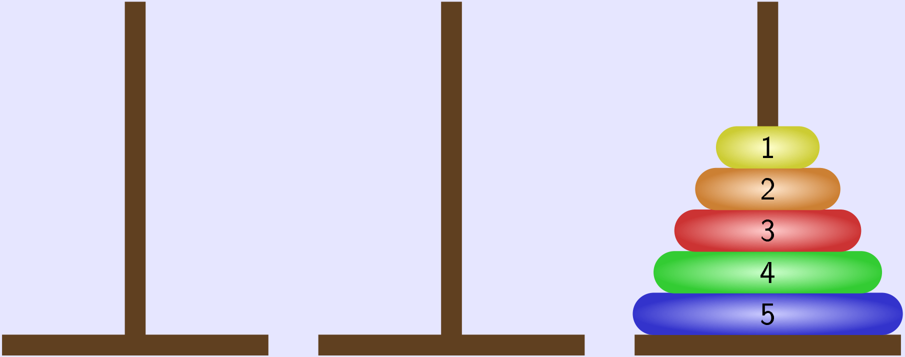
Moved disc from pole 2 to pole 1.

Torre de Hanoi



Moved disc from pole 2 to pole 3.

Torre de Hanoi



Moved disc from pole 1 to pole 3.

Torre de Hanoi

Torre de Hanoi

```java
class GFG
{
static void towerOfHanoi(int n, char from_rod,
                    char to_rod, char aux_rod)
{
  System.out.println("n = "+n);
    if (n == 1)
    {
        System.out.println("(Print 1, n==1) Move disk 1 from rod "+
                         from_rod+" to rod "+to_rod);
        return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod); // call 1
    System.out.println("(Print 2, n>1) Move disk "+ n + " from rod " +
                   from_rod +" to rod " + to_rod );
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);  // call 2
}

// Driver code
public static void main(String args[])
{
    int n = 4; // Number of disks
    towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
}
}
```

```
n = 4
n = 3
n = 2
n = 1
(Print 1, n==1) Move disk 1 from rod A to rod B
(Print 2, n>1) Move disk 2 from rod A to rod C
n = 1
(Print 1, n==1) Move disk 1 from rod B to rod C
(Print 2, n>1) Move disk 3 from rod A to rod B
n = 2
n = 1
(Print 1, n==1) Move disk 1 from rod C to rod A
(Print 2, n>1) Move disk 2 from rod C to rod B
n = 1
(Print 1, n==1) Move disk 1 from rod A to rod B
(Print 2, n>1) Move disk 4 from rod A to rod C
n = 3
n = 2
n = 1
(Print 1, n==1) Move disk 1 from rod B to rod C
(Print 2, n>1) Move disk 2 from rod B to rod A
n = 1
(Print 1, n==1) Move disk 1 from rod C to rod A
(Print 2, n>1) Move disk 3 from rod B to rod C
n = 2
n = 1
(Print 1, n==1) Move disk 1 from rod A to rod B
(Print 2, n>1) Move disk 2 from rod A to rod C
n = 1
(Print 1, n==1) Move disk 1 from rod B to rod C
```

Linhas no Plano

Linhas no Plano

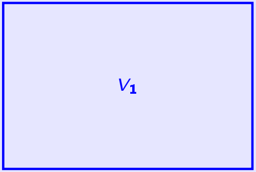
Problem

Popularly: How many slices of pizza can a person obtain by making n straight cuts with a pizza knife?

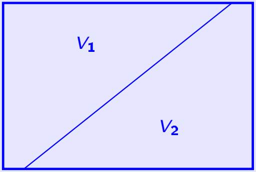
Academically: What is the maximum number L_n of regions defined by n lines in the plane?

Solved first in 1826, by the Swiss mathematician Jacob Steiner .

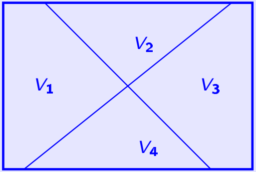
Linhas no Plano



$L_0 = 1$



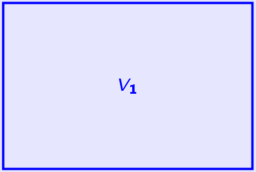
$L_1 = 2$



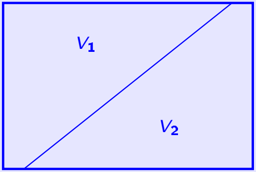
$L_2 = 4$

$L_3 = L_2 + 3 = 7$

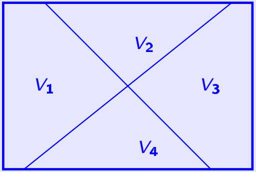
Linhas no Plano



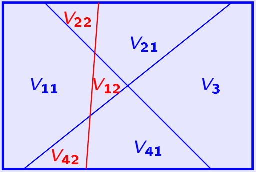
$$L_0 = 1$$



$$L_1 = 2$$



$$L_2 = 4$$



$$L_3 = L_2 + 3 = 7$$

Linhas no Plano

Observation:

The n-th line (for $n > 0$) increases the number of regions by k

iff it splits k of the "old regions"

iff it hits the previous lines in $k-1$ different places.

Linhas no Plano

Observation:

The n-th line (for $n > 0$) increases the number of regions by k

iff it splits k of the "old regions"

iff it hits the previous lines in $k-1$ different places.

Linhas no Plano

Observation:

The n-th line (for $n > 0$) increases the number of regions by k

iff it splits k of the "old regions"

iff it hits the previous lines in $k - 1$ different places.

Linhas no Plano

Observation:

The n-th line (for $n > 0$) increases the number of regions by k

iff it splits k of the "old regions"

iff it hits the previous lines in $k-1$ different places.

k must be less or equal to n. – Why?

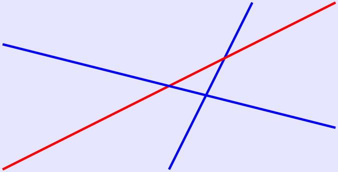
Linhas no Plano

Observation:

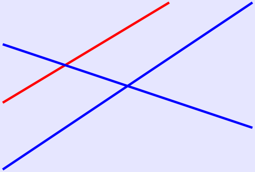
The n-th line (for $n > 0$) increases the number of regions by k

iff it splits k of the "old regions"

iff it hits the previous lines in $k-1$ different places.



$k = 3$; 2 places



$k = 2$; 1 places

Therefore the new line can intersect the $n-1$ "old" lines in at most "n- 1" different points, we have established the upper bound

$$\boxed{L_n \leqslant L_{n-1} + n \qquad \text{for } n > 0.}$$

If n-th line is not parallel to any of the others (hence it intersects them all), and doesn't go through any of the existing intersection points (hence it intersects them all in different places) then we get the recurrent equation:

$$L_0 = 1;$$
$$L_n = L_{n-1} + n \qquad \text{for } n > 0.$$

n	0	1	2	3	4	5	6	7	8	9	$\cdots$
L_n	1	2	4	7	11	16	22	29	37	46	$\cdots$

Therefore the new line can intersect the $n-1$ "old" lines in at most "n- 1" different points, we have established the upper bound

$$\boxed{L_n \leqslant L_{n-1} + n \qquad \text{for } n > 0.}$$

If n-th line is not parallel to any of the others (hence it intersects them all), and doesn't go through any of the existing intersection points (hence it intersects them all in different places) then we get the recurrent equation:

$$\begin{aligned} L_0 &= 1; \\ L_n &= L_{n-1} + n \qquad \text{for } n > 0. \end{aligned}$$

n	0	1	2	3	4	5	6	7	8	9	$\cdots$
L_n	1	2	4	7	11	16	22	29	37	46	$\cdots$

Linhas no Plano

Observation:

$$
\begin{aligned}
L_n &= L_{n-1} + n \\
&= L_{n-2} + (n-1) + n \\
&= L_{n-3} + (n-2) + (n-1) + n \\
&\cdots\cdots\cdots \\
&= L_0 + 1 + 2 + \ldots + (n-2) + (n-1) + n \\
&= 1 + S_n, \qquad \text{where } S_n = 1 + 2 + 3 + \ldots + (n-1) + n
\end{aligned}
$$

Evaluation of $S_n = 1 + 2 + \cdots + (n-1) + n$.

Recurrent equation:

$$
\begin{aligned}
S_0 &= 0; \\
S_n &= S_{n-1} + n \qquad \text{for } n > 0.
\end{aligned}
$$

Solution (Gauss, 1786):

$$
\begin{array}{ccccccccccc}
S_n & = & 1 & + & 2 & + & \cdots & + & (n-1) & + & n \\
+S_n & = & n & + & (n-1) & + & \cdots & + & 2 & + & 1 \\
\hline
2S_n & = & (n+1) & + & (n+1) & + & \cdots & + & (n+1) & + & (n+1)
\end{array}
$$

$$2S_n = n(n+1)$$

$$S_n = \frac{n(n+1)}{2}$$

Linhas no Plano

Evaluation of $S_n = 1 + 2 + \cdots + (n-1) + n$.

Recurrent equation:

$$
\begin{aligned}
S_0 &= 0; \\
S_n &= S_{n-1} + n \qquad\qquad \text{for } n > 0.
\end{aligned}
$$

Solution (Gauss, 1786):

$$
\begin{array}{ccccccccccc}
S_n & = & 1 & + & 2 & + & \cdots & + & (n-1) & + & n \\
+S_n & = & n & + & (n-1) & + & \cdots & + & 2 & + & 1 \\
\hline
2S_n & = & (n+1) & + & (n+1) & + & \cdots & + & (n+1) & + & (n+1)
\end{array}
$$

$$2S_n = n(n+1)$$

$$S_n = \frac{n(n+1)}{2}$$

Evaluation of $S_n = 1 + 2 + \cdots + (n-1) + n$.

Recurrent equation:

$$\begin{aligned} S_0 &= 0; \\ S_n &= S_{n-1} + n \qquad\qquad \text{for } n > 0. \end{aligned}$$

Solution (Gauss, 1786):

$$\begin{array}{ccccccccccc} S_n & = & 1 & + & 2 & + & \cdots & + & (n-1) & + & n \\ +S_n & = & n & + & (n-1) & + & \cdots & + & 2 & + & 1 \\ \hline 2S_n & = & (n+1) & + & (n+1) & + & \cdots & + & (n+1) & + & (n+1) \end{array}$$

$$2S_n = n(n+1)$$

$$S_n = \frac{n(n+1)}{2}$$

Theorem: Closed formula for L_n

$$L_n = \frac{n(n+1)}{2} + 1, \qquad \text{for } n > 0.$$

Proof (by induction).

Basis: $L_0 = \frac{0(0+1)}{2} + 1 = 1$.

Step: Let assume $L_n = \frac{n(n+1)}{2} + 1$ and evaluate

$$\begin{aligned} L_{n+1} &= L_n + n + 1 \\ &= \frac{n(n+1)}{2} + 1 + n + 1 \\ &= \frac{n(n+1) + 2 + 2n}{2} + 1 \\ &= \frac{n(n+1) + 2(n+1)}{2} + 1 \\ &= \frac{(n+1)(n+2)}{2} + 1. \end{aligned}$$

Q.E.D.

O problema de Josephus

O historiador judeu Flavius Josephus (37100) participou da revolta contra Roma no ano 66 e escapou do massacre após a captura da fortaleza em uma escura caverna, diz a lenda, que 41 rebeldes foram cercados por tropas romanas e antes de serem capturados, eles escolheram o suicídio em massa. Josephus e seu companheiro não pareciam muito convencidos do sacrifício, então ele propôs o seguinte: sentados em uma mesa circular, à partir de um certo escolhido, a terceira pessoa seria eliminada, até que apenas dois sobrevivessem. Josephus calculou as posições para que ele e seu companheiro sobrevivessem ao processo.

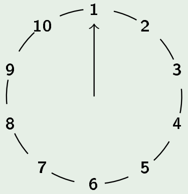
O problema de Josephus

Veremos a seguir o que talvez seja um dos primeiros problemas combinatórios da história, uma variação sobre o problema original de Josephus. Sabendo que há n pessoas numeradas de 1 a n em um círculo, eliminaremos cada segunda pessoa restante até sobrar um única pessoa. Estamos interessados em calcular J(n), o número do sobrevivente.

O problema de Josephus

Para entendermos melhor a questão veremos o que acontece quando $n = 10$. Após a primeira volta, eliminamos nessa ordem as pessoas de número 2,4,6,8 e 10.
Na segunda volta descartamos 3 e 7. E finalmente, na última volta, eliminamos 1 e 9, restando somente a pessoa de número 5.

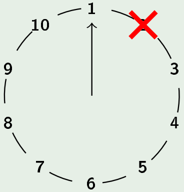
Example, $n = 10$.



The elimination order is
2, 4, 6, 8, 10, 3, 7, 1, 9 .

So, we have $J(10) = 5$

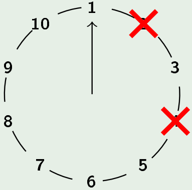
Example, $n = 10$.



The elimination order is 2, 4, 6, 8, 10, 3, 7, 1, 9 .

So, we have $J(10) = 5$

Example, $n = 10$.

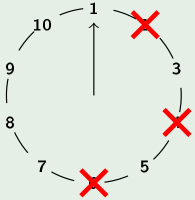


The elimination order is

2, 4, 6, 8, 10, 3, 7, 1, 9 .

So, we have $J(10) = 5$

Example, $n = 10$.

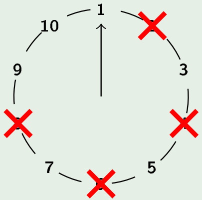


The elimination order is

2, 4, 6, 8, 10, 3, 7, 1, 9.

So, we have $J(10) = 5$

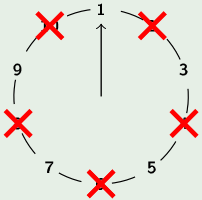
Example, $n = 10$.



The elimination order is
2, 4, 6, 8, 10, 3, 7, 1, 9.

So, we have $J(10) = 5$

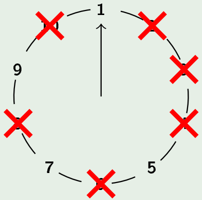
Example, $n = 10$.



The elimination order is
2, 4, 6, 8, 10, 3, 7, 1, 9.

So, we have $J(10) = 5$

Example, $n = 10$.

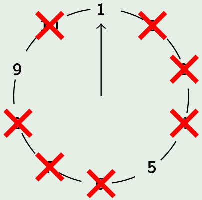


The elimination order is

2, 4, 6, 8, 10, 3, 7, 1, 9.

So, we have $J(10) = 5$

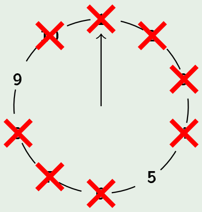
Example, $n = 10$.



The elimination order is
2, 4, 6, 8, 10, 3, 7, 1, 9.

So, we have $J(10) = 5$

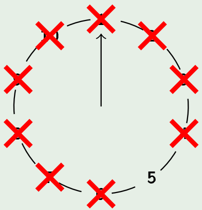
Example, $n = 10$.



The elimination order is
2, 4, 6, 8, 10, 3, 7, 1, 9.

So, we have $J(10) = 5$

Example, $n = 10$.



The elimination order is
2, 4, 6, 8, 10, 3, 7, 1, 9.

So, we have $J(10) = 5$

O problema de Josephus

Observamos que $J(n)$ é sempre ímpar, o que é coerente com o problema, uma vez que, como foi observado para $n = 10$, na primeira volta eliminamos todos que possuem número par.

O problema de Josephus

Logo, $J(2n+1) = 2J(n)+1,\ n \geqslant 1$. Combinando estas equações com sua condição inicial, chegamos à nossa relação de recorrência:

$$\begin{aligned} J(1) &= 1; \\ J(2n) &= 2J(n) - 1, \quad para \quad n \geqslant 1; \\ J(2n+1) &= 2J(n) + 1, \quad para \quad n \geqslant 1. \end{aligned} \tag{3.5}$$

A partir desta relação de recorrência podemos construir a seguinte tabela para valores pequenos de n:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	⋯
$J(n)$	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15	1	⋯

Figura 3.31: Tabela de eliminação

O problema de Josephus

Percebemos que a cada potência de 2 em n é formado um grupo que sempre se inicia com $J(n) = 1$ e, à medida em que n cresce, $J(n)$ aumenta de 2 em 2 dentro desse grupo. Então se escrevermos n na forma $n = 2^m + l$, em que 2^m é a maior potência de 2 que não é maior que n, e l é o que sobrou, teremos:

$$J(2^m + l) = 2l + 1, \quad m \geqslant 0 \;\; e \;\; 0 \leqslant l < 2^m \tag{3.6}$$

Recorrência e Recursividade

- A ideia de recursividade é a de um processo que é definido a partir de si próprio. No caso de um algoritmo, esse é definido invocando a si mesmo.
- Uma recorrência, ou relação de recorrência, é uma expressão que dá o valor de uma função num dado "ponto"em termos dos valores da mesma função em "pontos anteriores".

Formal Uma relação de recorrência ou, como também é chamada, uma equação de recorrência, é uma relação que determina cada termo de uma dada sequência, a partir de certo termo, em função dos termos anteriores

Recorrência

- Resolver uma recorrência é encontrar uma "fórmula fechada"que dê o valor da função diretamente em termos de seu argumento (e sem subexpressões da forma "+ ... +"ou contendo "$\sum$"ou "$\prod$"). Tipicamente, a fórmula fechada é uma combinação de polinômios, quocientes de polinômios, logaritmos, exponenciais, etc.
- Para analisar o consumo de tempo de um algoritmo recursivo é necessário resolver uma recorrência.
- Existem casos complexos onde as recorrências não admitem fórmula fechada. Nestes casos, já é suficiente calcular uma cota superior (Upper Bound).
- A fórmula fechada do problema das linhas no plano é uma cota superior.

Classificação das Relações de Recorrência

- Uma equação de recorrência na qual cada termo depende exclusivamente dos anteriores é dita homogênea.
- Se, além dos termos anteriores, cada elemento da sequência está também em função de um termo independente da sequência, a recorrência é dita não-homogênea.
- Uma relação de recorrência é dita linear quando a função que relaciona cada termo aos termos anteriores é linear.
- Além disso, é dita de primeira ordem quando cada termo da sequência é obtido a partir do termo imediatamente anterior a ele, ou seja, quando a_n está em função de $an - 1$

Resolvendo recorrência linear homogenea de primeira ordem

- Uma relação de recorencia linear homogenea de primeira ordem é:
  a_{n+1}=g(n) a_n
  onde g(n) e a_n são não nulos e g(n) é uma função linear.

Podemos então escrever:

$$a_2 = g(1)a_1$$
$$a_3 = g(2)a_2$$
$$a_4 = g(3)a_3$$
$$\vdots$$
$$a_{n+1} = g(n)a_n$$

Substituindo cada termo na expressão seguinte, obtemos:

$$a_{n+1} = a_1 \cdot g(1) \cdot g(2) \cdot g(3) \cdot ... \cdot g(n),$$

ou seja,

$$a_{n+1} = a_1 \prod_{j=1}^{n} g(j) \ .$$

Classifique a recorrência

- Sequência de Fibonacci
- $F_1 = F_2 = 1$
- $F_n = F_{n-1} + Fn - 2$

Classifique!

Classifique a recorrência

- Sequência de Fibonacci
- $F_1 = F_2 = 1$
- $F_n = F_{n-1} + F_{n-2}$

É uma recorrência Homogenea, Linear e de Segunda Ordem.

Vídeos de Apoio

Introdução de Recorrências `https://youtu.be/S9zDoNAr3d4`
Torre de Hanoi `https://youtu.be/KGidiq5vh98`
Josephus `https://youtu.be/beFd1_AEKW4`
Recorrências de Primeira Ordem `https://youtu.be/DtUO872ajTM`
Recorrências e Fórmulas Fechadas `https://youtu.be/IT679ay8Y2s`

Exercício

1) Escreve o conceito de recorrência e recursividade?
2) Quais classificações existem para relações de recorrência?
3) O que significa solucionar uma recorrência?
4) Toda recorrência admite uma fórmula fechada?
5) Cite 2 problemas que possuem recorrência linear homogênea de primeira ordem.
6) Cite 2 problemas que possuem recorrência linear não-homogênea de primeira ordem.
7) Cite 2 problemas que possuem recorrência linear homogênea de segunda ordem.
8) Como o conceito de cota superior pode ser aplicado em solução de problemas recorrentes?
9) Pesquise sobre o teorema Mestre e escreva resumidamente o que é e como ele pode ser aplicado.
10) Pesquise e escreva sobre uma variação do problema da torre de Hanoi.

Matemática Concreta

Problemas Recorrentes

Dr. A. Riker
Universidade Federal do Pará (UFPA)
afr@ufpa.br

2021.PL03