



# **Busca Tabu para N-Rainhas**

## **TRABALHO 1**

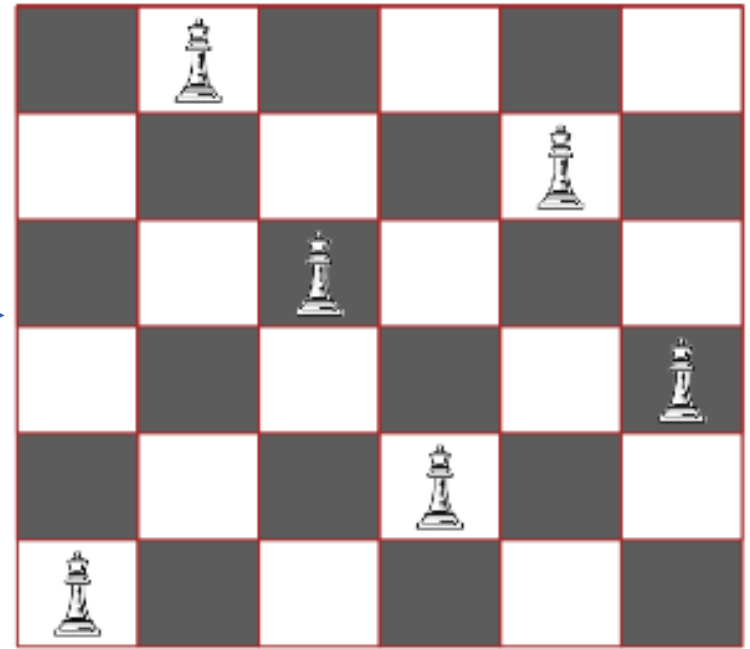
**Professor Ciniro Nametala  
Inteligência Artificial  
Instituto Federal de Minas Gerais**

# O problema das N-Rainhas

## Descrição

O problema das N-Rainhas consiste em encontrar uma disposição de  $n$  rainhas do jogo de xadrez em um tabuleiro  $n \times n$  de tal modo que nenhuma rainha ataque as outras de acordo com as regras do jogo. Uma rainha ataca outra quando elas estão posicionadas na mesma linha, coluna ou diagonal.

Esta é uma solução possível para o problema considerando um tabuleiro com 6 rainhas, logo 6 linhas e 6 colunas. Ou seja, com  $n$  igual a 6.

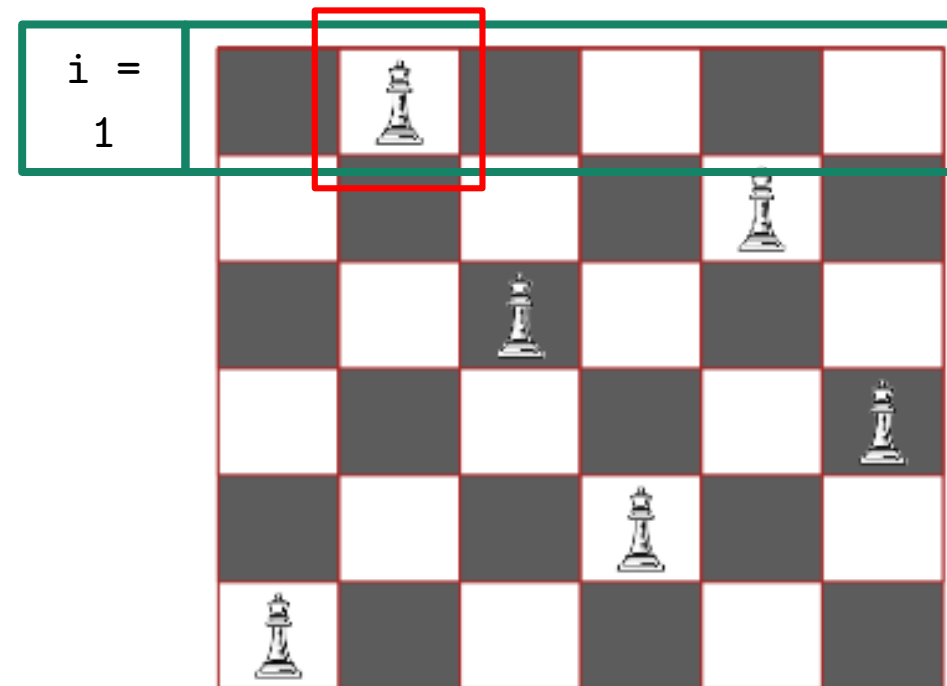
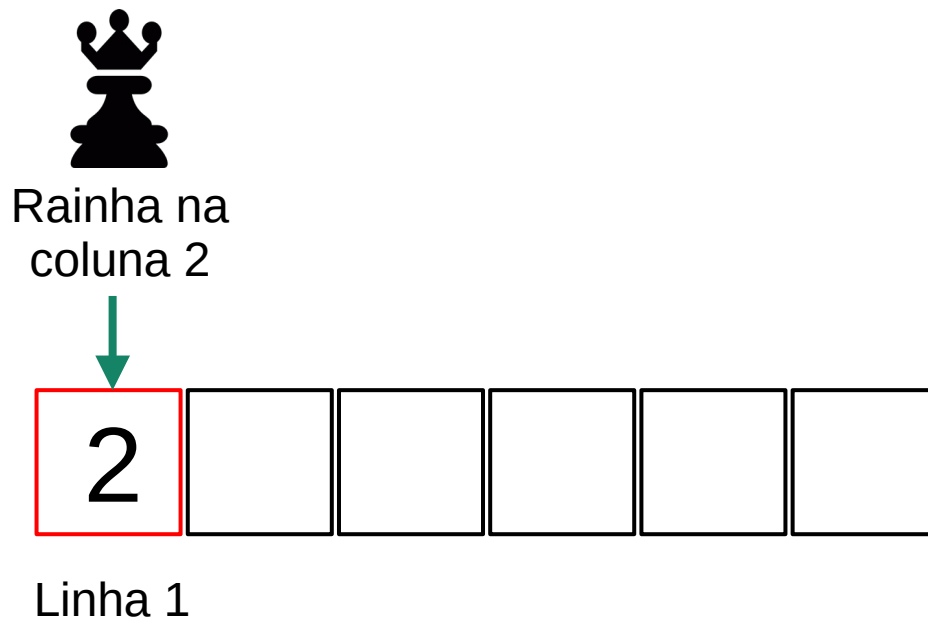


# O problema das N-Rainhas

## Representação da solução

Uma solução para o problema pode ser representada por um vetor  $s$  de  $n$  posições, em que cada célula  $S_i$  representa a coluna  $j$  em que a rainha da linha  $i$  está posicionada. Portanto, o par  $(i, S_i) = (i, j)$  representa a posição de uma rainha.

A disposição das rainhas em um problema  $n = 6$  com essa solução pode ser representada pelo vetor  $s = \{2, 5, 3, 6, 4, 1\}$  pois...

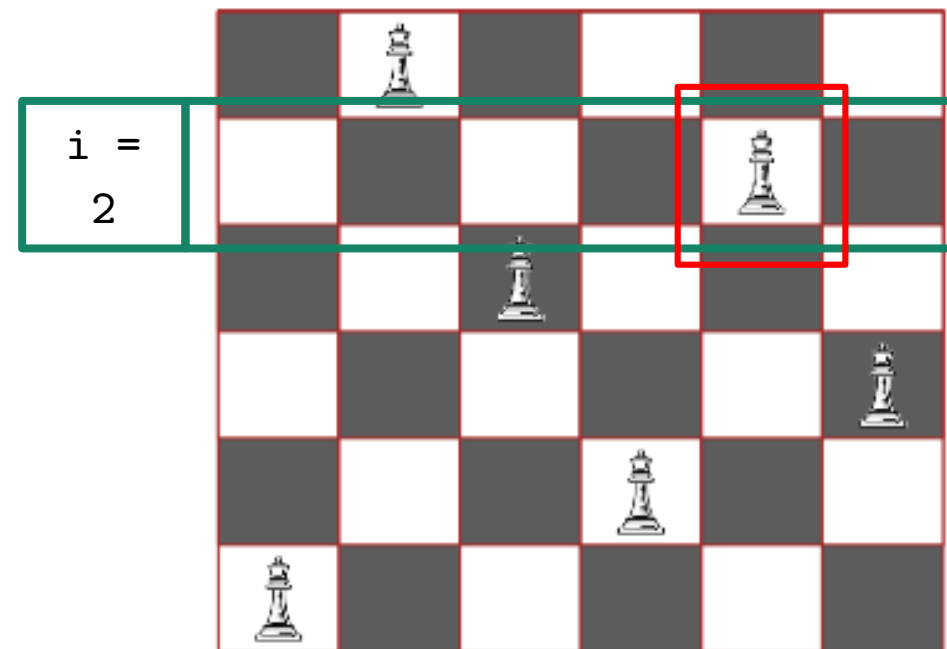
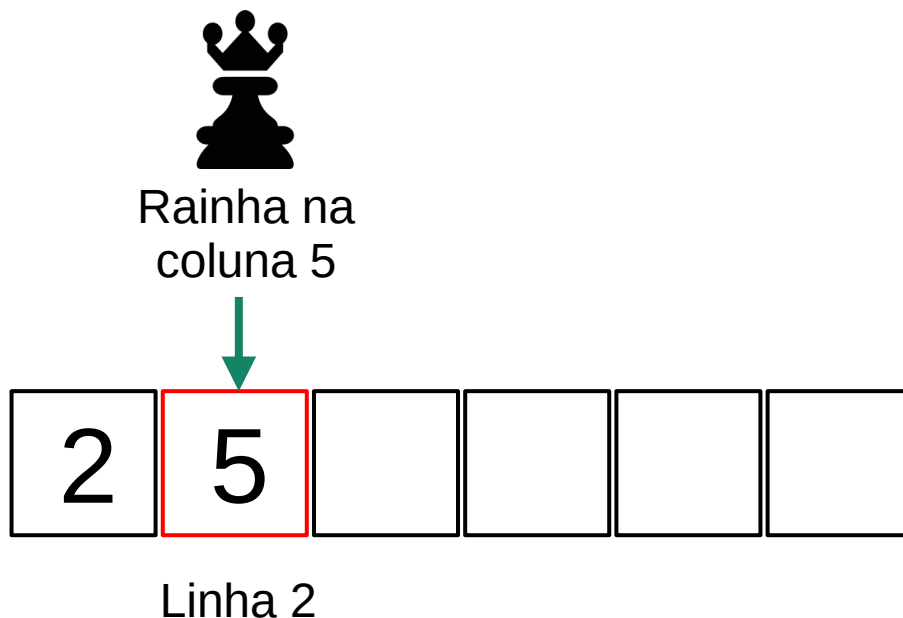


# O problema das N-Rainhas

## Representação da solução

Uma solução para o problema pode ser representada por um vetor  $s$  de  $n$  posições, em que cada célula  $S_i$  representa a coluna  $j$  em que a rainha da linha  $i$  está posicionada. Portanto, o par  $(i, S_i) = (i, j)$  representa a posição de uma rainha.

A disposição das rainhas em um problema  $n = 6$  com essa solução pode ser representada pelo vetor  $s = \{2, 5, 3, 6, 4, 1\}$  pois...



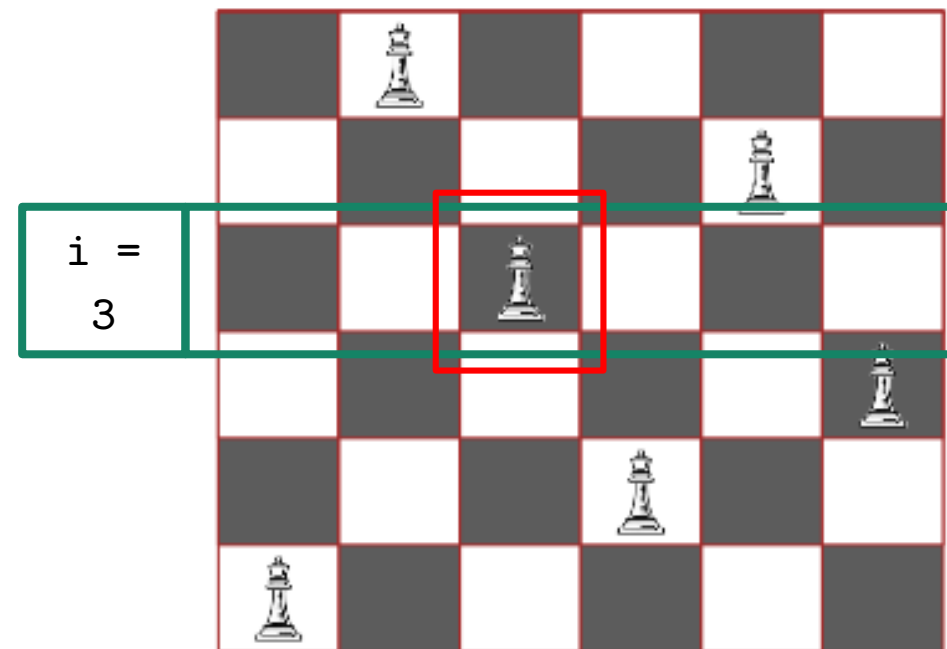
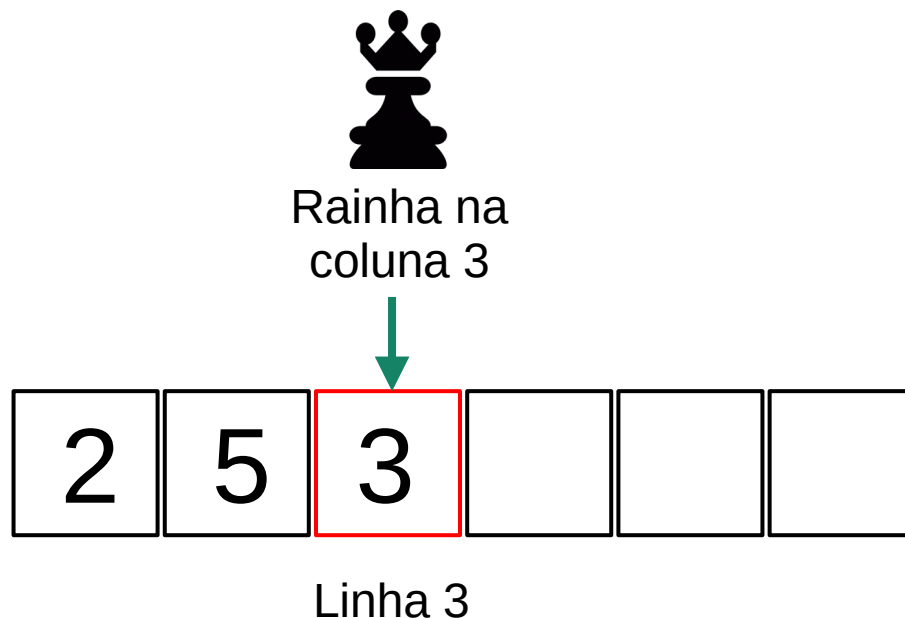


# O problema das N-Rainhas

## Representação da solução

Uma solução para o problema pode ser representada por um vetor  $s$  de  $n$  posições, em que cada célula  $S_i$  representa a coluna  $j$  em que a rainha da linha  $i$  está posicionada. Portanto, o par  $(i, S_i) = (i, j)$  representa a posição de uma rainha.

A disposição das rainhas em um problema  $n = 6$  com essa solução pode ser representada pelo vetor  $s = \{2, 5, 3, 6, 4, 1\}$  pois...

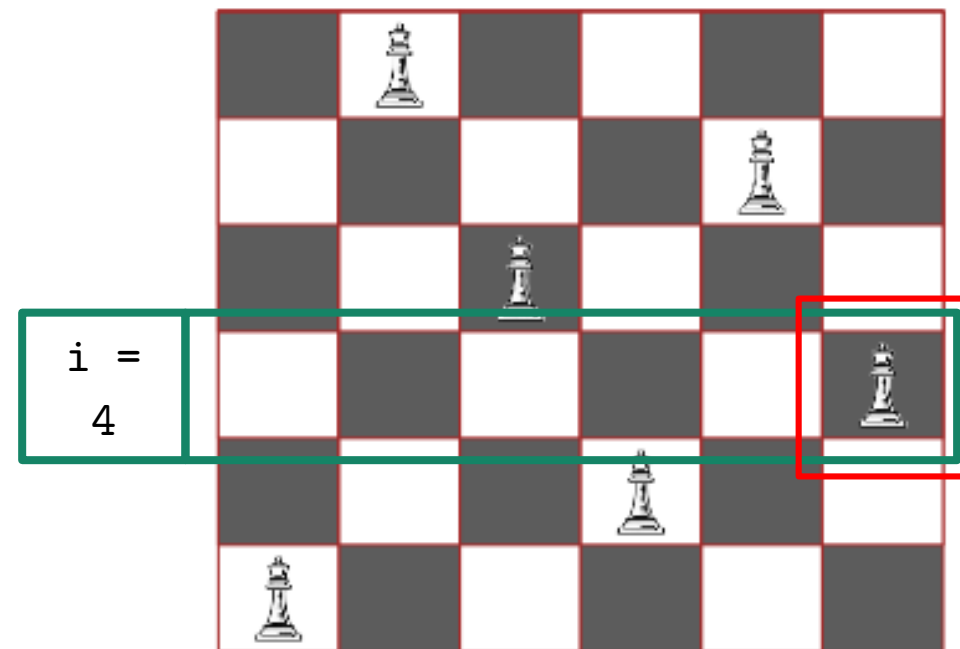
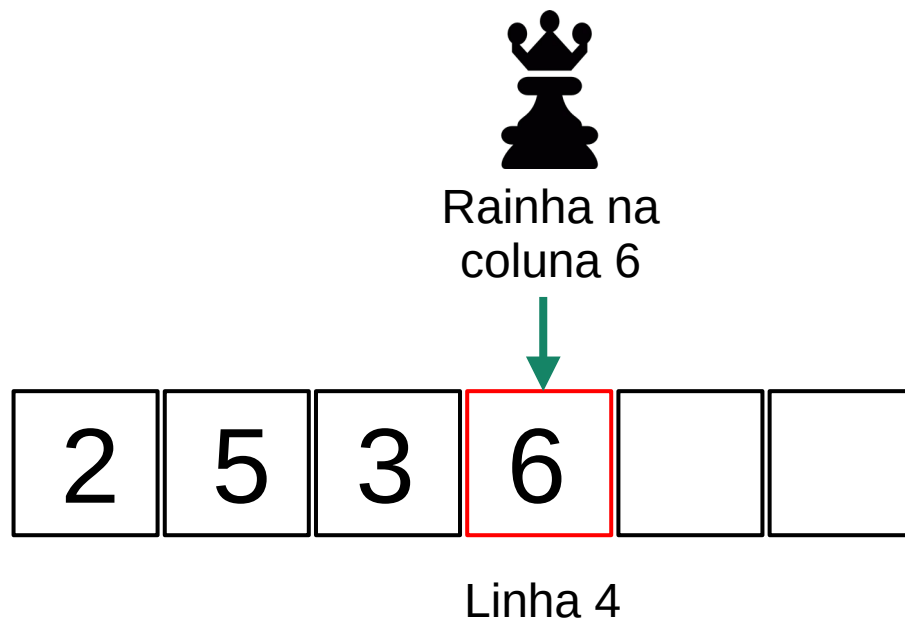


# O problema das N-Rainhas

## Representação da solução

Uma solução para o problema pode ser representada por um vetor  $s$  de  $n$  posições, em que cada célula  $S_i$  representa a coluna  $j$  em que a rainha da linha  $i$  está posicionada. Portanto, o par  $(i, S_i) = (i, j)$  representa a posição de uma rainha.

A disposição das rainhas em um problema  $n = 6$  com essa solução pode ser representada pelo vetor  $s = \{2, 5, 3, 6, 4, 1\}$  pois...

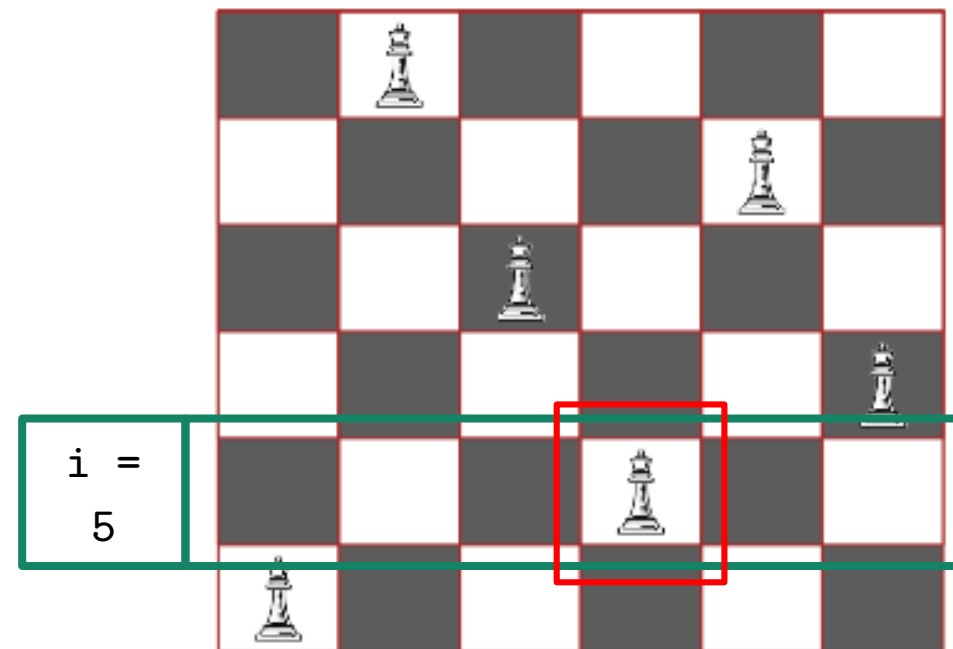
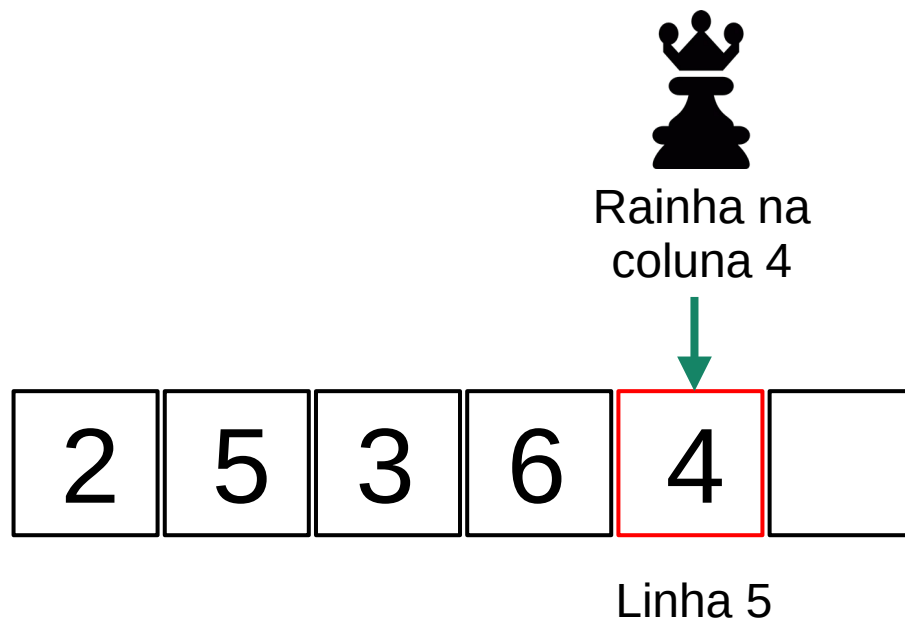


# O problema das N-Rainhas

## Representação da solução

Uma solução para o problema pode ser representada por um vetor  $s$  de  $n$  posições, em que cada célula  $S_i$  representa a coluna  $j$  em que a rainha da linha  $i$  está posicionada. Portanto, o par  $(i, S_i) = (i, j)$  representa a posição de uma rainha.

A disposição das rainhas em um problema  $n = 6$  com essa solução pode ser representada pelo vetor  $s = \{2, 5, 3, 6, 4, 1\}$  pois...

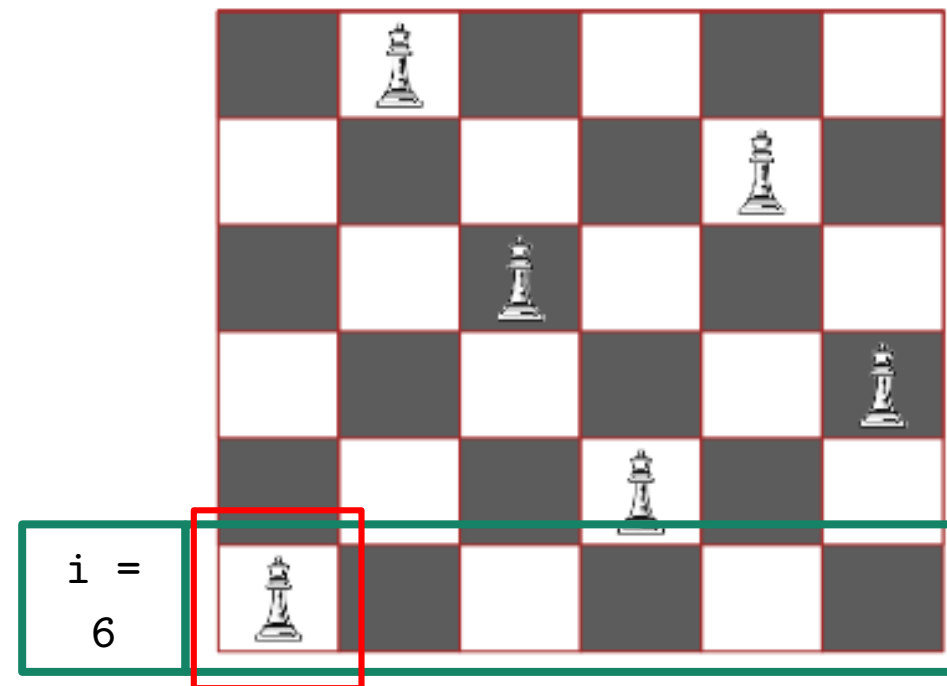
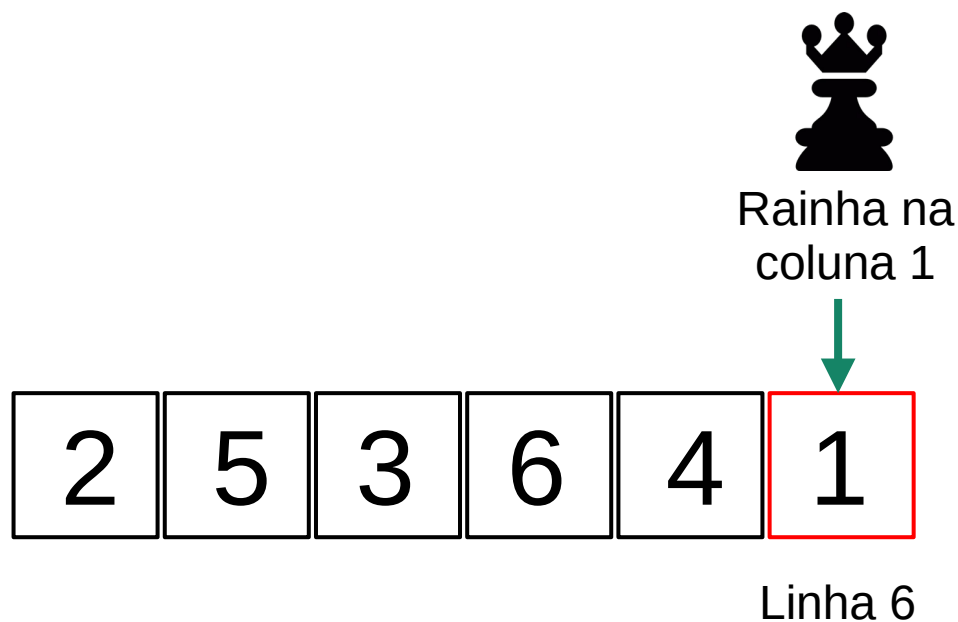


# O problema das N-Rainhas

## Representação da solução

Uma solução para o problema pode ser representada por um vetor  $s$  de  $n$  posições, em que cada célula  $S_i$  representa a coluna  $j$  em que a rainha da linha  $i$  está posicionada. Portanto, o par  $(i, S_i) = (i, j)$  representa a posição de uma rainha.

A disposição das rainhas em um problema  $n = 6$  com essa solução pode ser representada pelo vetor  $s = \{2, 5, 3, 6, 4, 1\}$  pois...





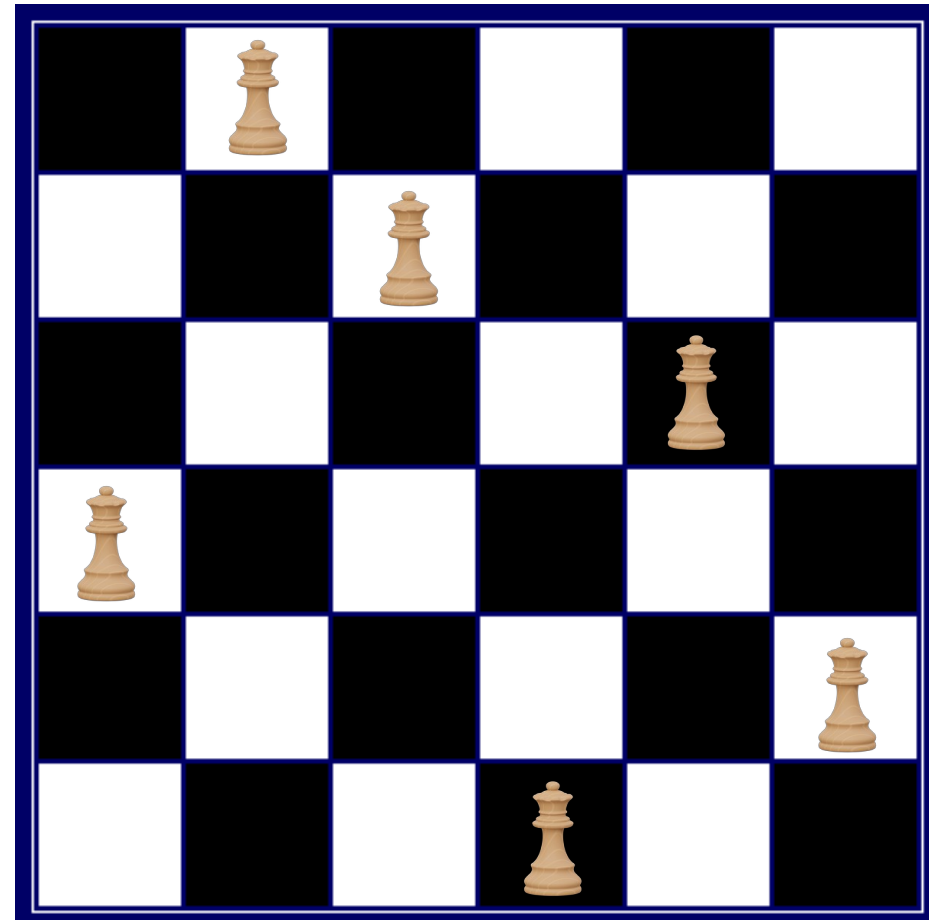
# O problema das N-Rainhas

## Computação de conflitos entre rainhas

Considere agora a seguinte configuração de rainhas:

2	3	5	1	6	4
---	---	---	---	---	---

Essa solução  
é boa?



# O problema das N-Rainhas

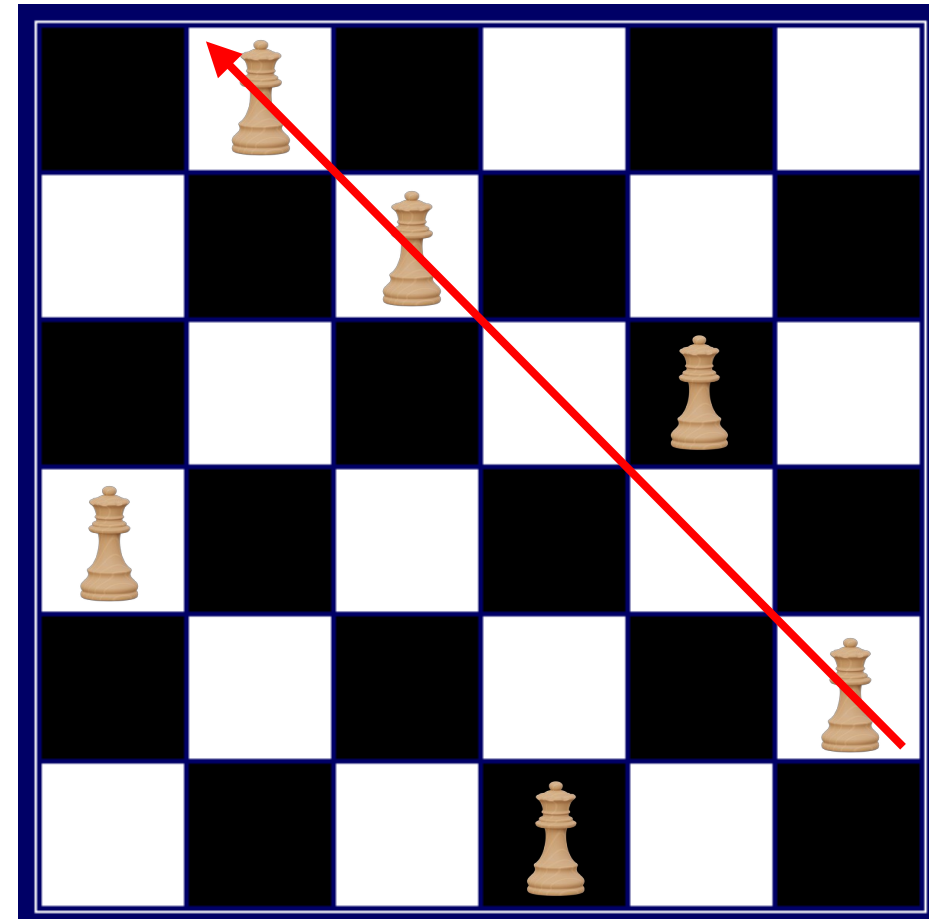
## Computação de conflitos entre rainhas

Considere agora a seguinte configuração de rainhas:

2	3	5	1	6	4
---	---	---	---	---	---

**2 rainhas** além do esperado nesta diagonal.

Não é.



# O problema das N-Rainhas

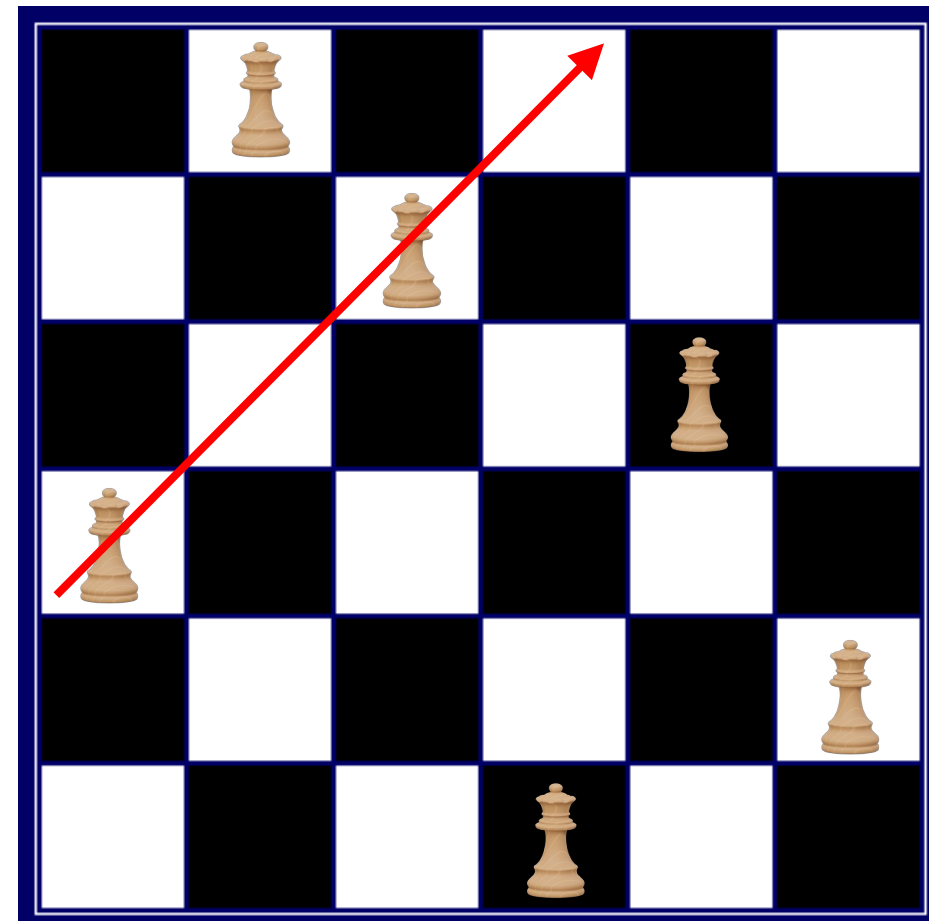
## Computação de conflitos entre rainhas

Considere agora a seguinte configuração de rainhas:

2	3	5	1	6	4
---	---	---	---	---	---

**1 rainha** além do  
esperado nesta diagonal.

Não é.



# O problema das N-Rainhas

## Computação de conflitos entre rainhas

Qual método usar para calcular a **quantidade de conflitos** usando o nosso vetor que representa uma solução?

2	3	5	1	6	4
---	---	---	---	---	---

# O problema das N-Rainhas

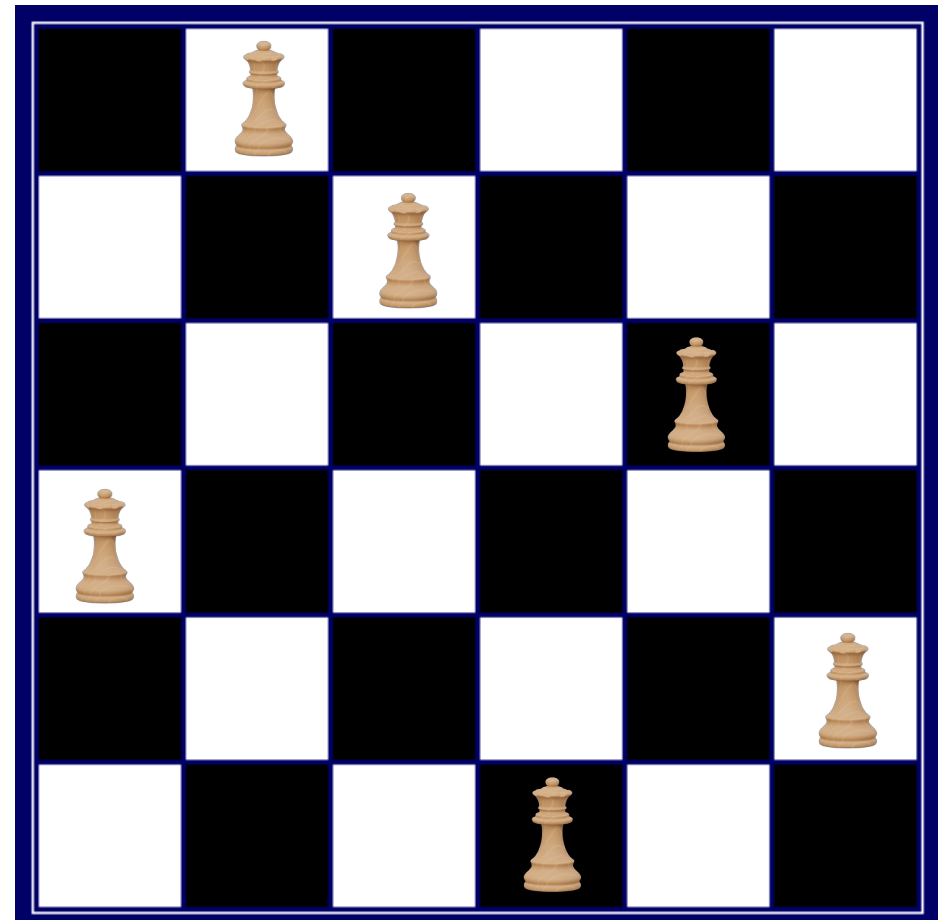
## Computação de conflitos entre rainhas

Primeiramente perceba que com esta representação, mesmo uma solução ruim como a do exemplo nunca gerará conflitos nas linhas e colunas.

2	3	5	1	6	4
---	---	---	---	---	---

Nesta representação se houverem conflitos, estes acontecerão apenas nas diagonais.

Vamos analisar as diagonais melhor.





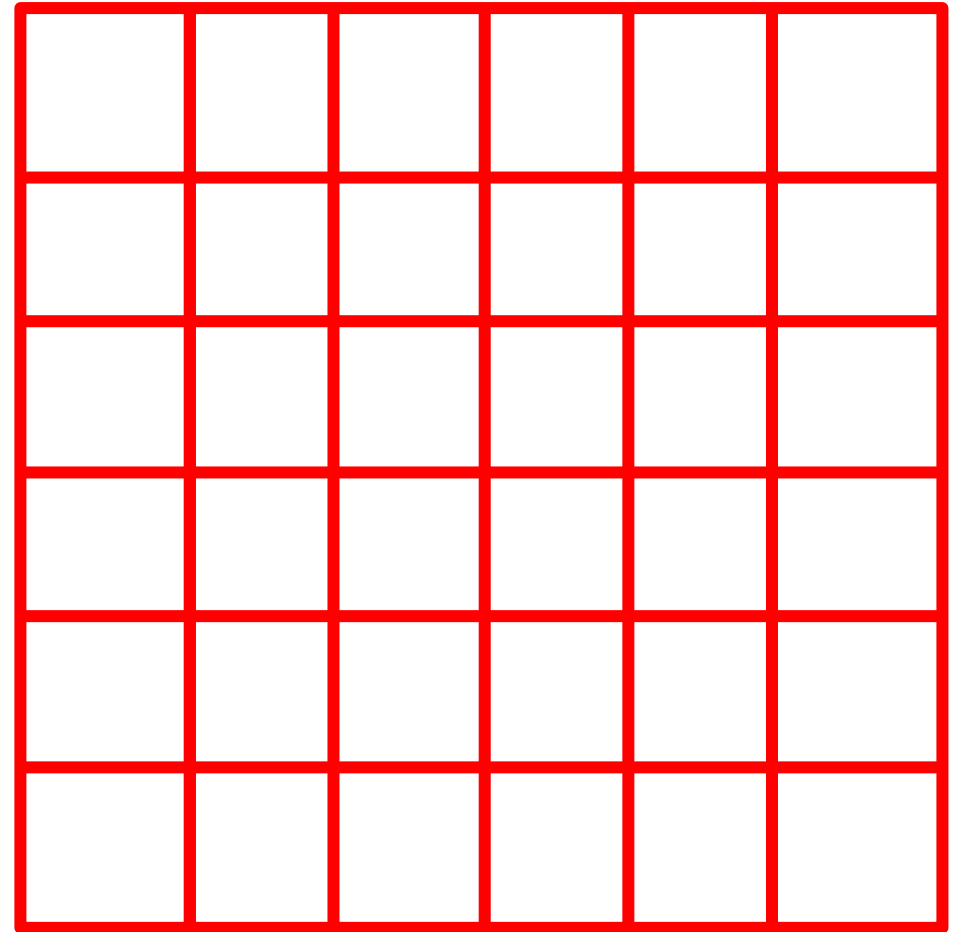
# O problema das N-Rainhas

## Computação de conflitos entre rainhas

Vamos considerar que o tabuleiro está dividido em duas diagonais:

(a): Diagonal positiva

(b): Diagonal negativa



# O problema das N-Rainhas

## Diagonal positiva

A **diagonal positiva (dp)** diz respeito a  $(2n - 1)$  índices que representam cada linha diagonal no tabuleiro.

Os limites dos índices na **dp** são  $[1 - n, n - 1]$ , portanto, neste caso de  $n = 6$ , temos 11 índices dentro de um intervalo igual a  $[-5, 5]$ .

Para se descobrir em qual diagonal está posicionada uma dada rainha nas **dps**, basta calcular o índice:

$$k = i - S_i$$

onde  $i$  é a linha e  $S_i$  a coluna em que ela está posicionada.

0	-1	-2	-3	-4	-5
1	0	-1	-2	-3	-4
2	1	0	-1	-2	-3
3	2	1	0	-1	-2
4	3	2	1	0	-1
5	4	3	2	1	0

# O problema das N-Rainhas

## Diagonal negativa

A **diagonal negativa** ( $dn$ ) diz respeito a  $(2n - 1)$  índices que representam cada linha diagonal no tabuleiro.

Os limites dos índices na  $dn$  são  $[2, 2n]$ , portanto, neste caso de  $n = 6$ , temos 11 índices dentro de um intervalo igual a  $[2, 12]$ .

Para se descobrir em qual diagonal está posicionada uma dada rainha nas  $dns$ , basta calcular o índice:

$$k = i + S_i$$

onde  $i$  é a linha e  $S_i$  a coluna em que ela está posicionada.

2	3	4	5	6	7
3	4	5	6	7	8
4	5	6	7	8	9
5	6	7	8	9	10
6	7	8	9	10	11
7	8	9	10	11	12

# O problema das N-Rainhas

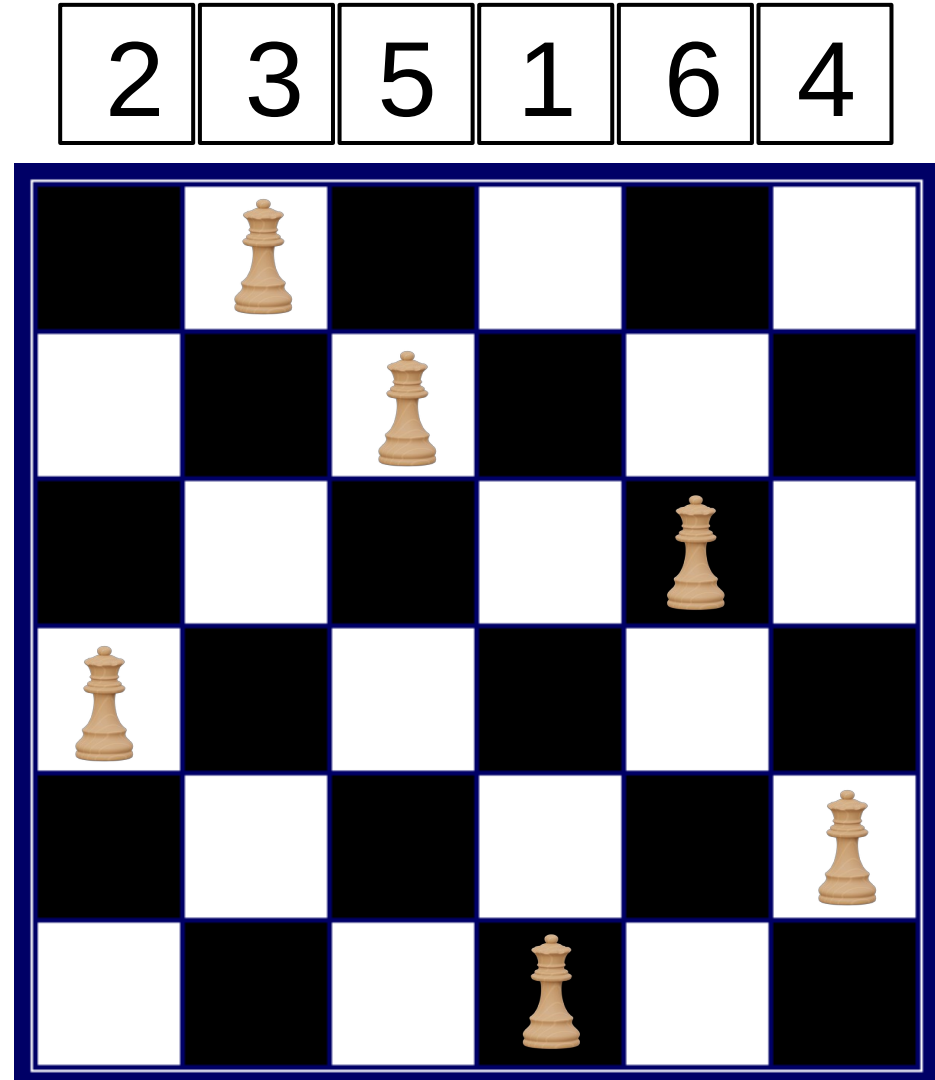
## Identificando a posição das rainhas nas diagonais

Considerando-se que as posições são determinadas por:

- **dp**:  $k = i - S_i$
- **dn**:  $k = i + S_i$

Temos que:

Rainha	i	$S_i$	dp(k)	dn(k)
1	1	2	-1	3
2	2	3	-1	5
3	3	5	-2	8
4	4	1	3	5
5	5	6	-1	11
6	6	4	2	10









# O problema das N-Rainhas

## Identificando a posição das rainhas nas diagonais

Perceba agora, segundo a tabela, onde estão posicionadas as rainhas segundo a **diagonal positiva**:

Rainha	i	$S_i$	$dp(k)$	$dn(k)$
1	1	2	-1	3
2	2	3	-1	5
3	3	5	-2	8
4	4	1	3	5
5	5	6	-1	11
6	6	4	2	10

2	3	5	1	6	4
0	-1 	-2	-3	-4	-5
1	0	-1 	-2	-3	-4
2	1	0	-1	-2 	-3
3 	2	1	0	-1	-2
4	3	2	1	0	-1 
5	4	3	2 	1	0









# O problema das N-Rainhas

## Identificando a posição das rainhas nas diagonais

A mesma coisa agora, segundo a **diagonal negativa**:

Rainha	i	$S_i$	$dp(k)$	$dn(k)$
1	1	2	-1	3
2	2	3	-1	5
3	3	5	-2	8
4	4	1	3	5
5	5	6	-1	11
6	6	4	2	10

2	3	5	1	6	4
2		4	5	6	7
3	4		6	7	8
4	5	6	7		9
	6	7	8	9	10
6	7	8	9	10	
7	8	9		11	12

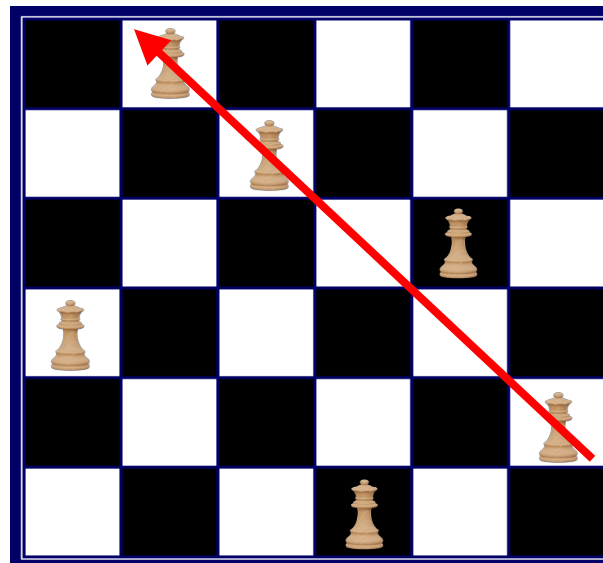
# O problema das N-Rainhas

## Contagem de conflitos

Identificado agora o posicionamento nas diagonais e sabendo que só podem existir conflitos entre diagonais, então basta contar a quantidade de rainhas que estão ocupando diagonais iguais indevidamente:

Rainha	dp(k)	dn(k)
1	-1	3
2	-1	5
3	-2	8
4	3	5
5	-1	11
6	2	10

**2 rainhas** além do esperado na **diagonal positiva** de índice -1.



0	-1	-2	-3	-4	-5
1	0	-1	-2	-3	-4
2	1	0	-1	-2	-3
3	2	1	0	-1	-2
4	3	2	1	0	-1
5	4	3	2	1	0

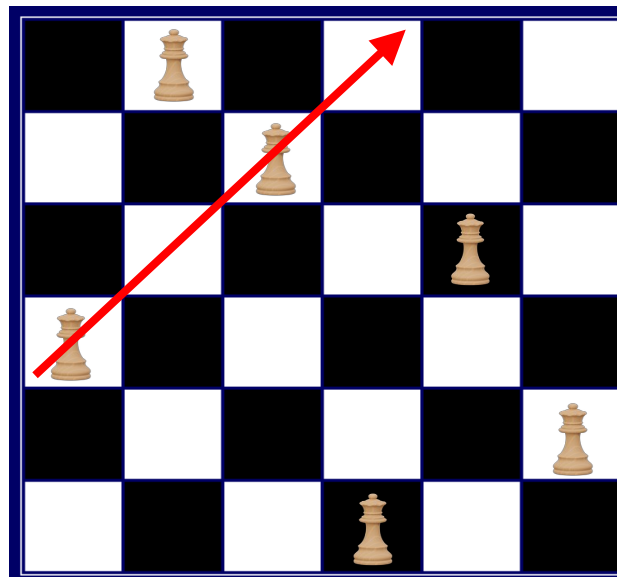
# O problema das N-Rainhas

## Contagem de conflitos

Identificado agora o posicionamento nas diagonais e sabendo que só podem existir conflitos entre diagonais, então basta contar a quantidade de rainhas que estão ocupando diagonais iguais indevidamente:

Rainha	dp(k)	dn(k)
1	-1	3
2	-1	5
3	-2	8
4	3	5
5	-1	11
6	2	10

**1 rainha** além do esperado na **diagonal negativa** de índice 5.



2	3	4	5	6	7
3	4	5	6	7	8
4	5	6	7	8	9
5	6	7	8	9	10
6	7	8	9	10	11
7	8	9	10	11	12

# O problema das N-Rainhas

## Contagem de conflitos

Considerando a contagem em ambas as diagonais percebemos que existem pelo menos **3 rainhas** em posições indevidas com essa alternativa de solução, considerando que pode existir apenas uma por diagonal:

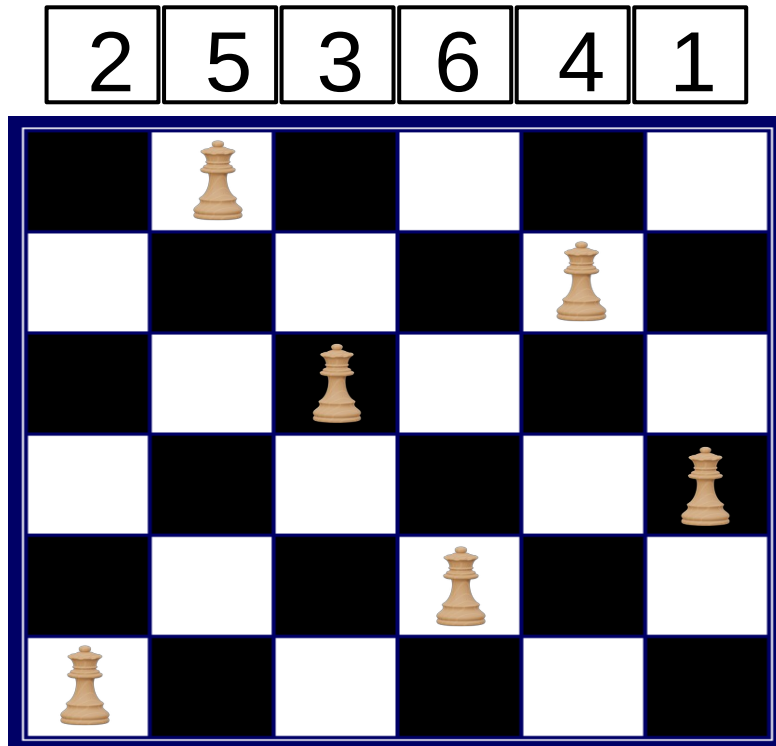
Rainha	dp(k)	dn(k)
1	-1	3
2	-1	5
3	-2	8
4	3	5
5	-1	11
6	2	10

2	3	5	1	6	4
---	---	---	---	---	---

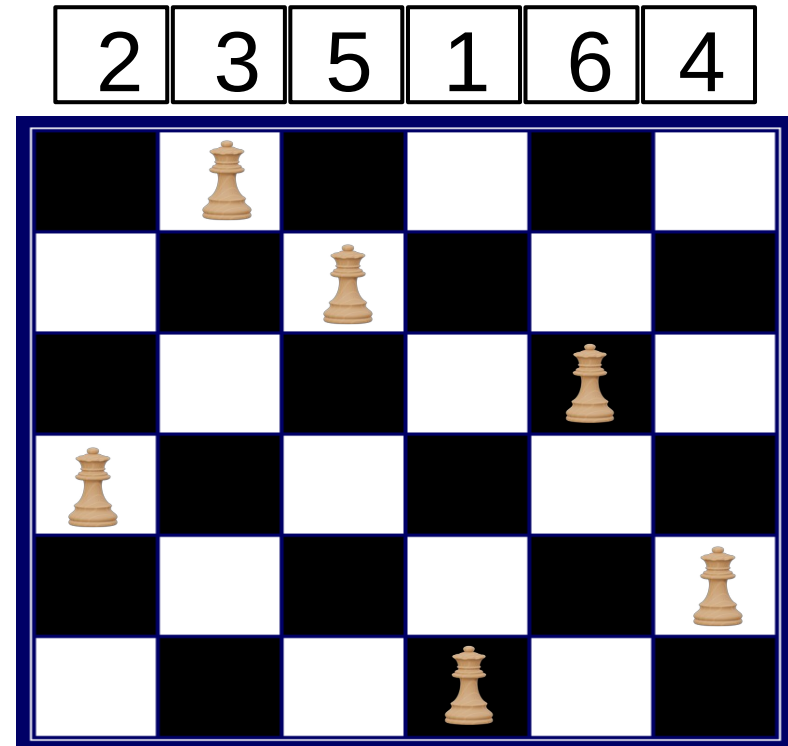
# O problema das N-Rainhas

## Fitness e Função Objetivo

A solução ótima nesse caso é qualquer solução em que existam **zero conflitos**. Nos chamamos esse valor que define a qualidade da solução de **custo ou fitness**, e a função para calculá-lo de **função objetivo, de perda ou de custo**.



Fitness dessa solução: 1



Fitness dessa solução: 3

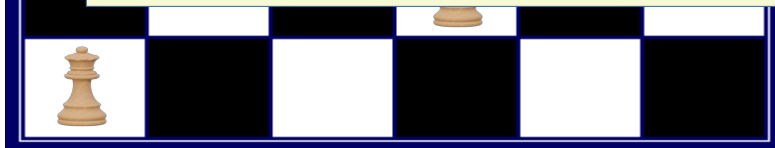


# O problema das N-Rainhas

## Fitness e Função Objetivo

A solução ótima nesse caso é qualquer solução em que existam **zero conflitos**. Nos chamamos esse valor que define a qualidade da solução de **custo ou fitness**, e a função para calculá-lo é a **função de fitness**.

Esse é um **problema de minimização**, ou seja, queremos gerar soluções que tenham o **menor valor de fitness possível**. Pois, valores pequenos neste caso equivalem a menos conflitos entre rainhas – que é exatamente o que buscamos encontrar.



Fitness dessa solução: 0



Fitness dessa solução: 3

# O problema das N-Rainhas

## O tamanho do problema

Testar todas as possíveis posições das rainhas é uma tarefa quase impossível quando você aumenta o tamanho do tabuleiro. Veja...

Um tabuleiro 8x8 tem 64 posições. Para a primeira rainha temos 64 possibilidades, para a segunda 63, para a terceira 62 e assim por diante. Logo temos  $64 \cdot 63 \cdot 62 \cdot 61 \cdot 60 \cdot 59 \cdot 58 \cdot 57 / 8! = C_{64,8}$ , ou seja, o número de combinações de 64 elementos tomados 8 a 8.

$$C_{64,8} = 64! / (8! \cdot (64-8)!) = 64! / (8! \cdot 56!) = 4.426.165.368.$$

Se você dobrar o tamanho do tabuleiro isso sobe para  $1.4 \times 10^{12}$  e assim por diante.

**É um problema  $O(n!)$**

# O problema das N-Rainhas

## O tamanho do problema

Então como resolver?

Hoje vamos aprender uma forma  
muito eficaz com a nossa primeira IA:  
O algoritmo Busca Tabu.



# Busca Tabu

## Definição

A Busca Tabu (*Tabu Search* – TS) é um algoritmo que categorizamos como metaheurística de busca local. O algoritmo foi criado em 1977 por Glover e Laguna.

O algoritmo pode ser resumido segundo o pseudocódigo a seguir:

1. Considere uma **solução inicial** qualquer como a **solução atual**
2. Avalie elementos na **vizinhança** da **solução atual** levando em conta movimentos que **não sejam tabu** ou que **atendam** a algum **critério de aspiração**.
3. Tome a **melhor solução vizinha** como a **solução atual**
4. Atualize a **lista tabu**
5. Se o **critério de parada** não for satisfeito, volte ao passo 2, senão termine o algoritmo.

# Busca Tabu

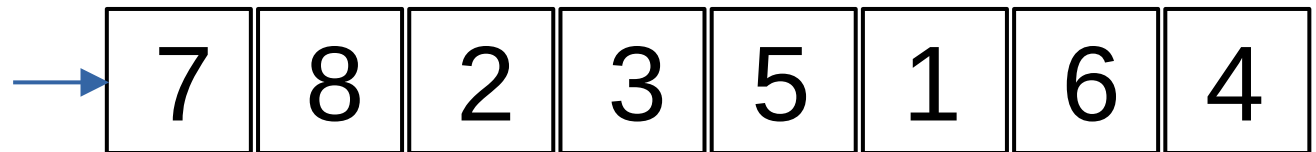
## Passo 1 – Solução inicial

1. Considere uma **solução inicial** qualquer como a **solução atual**

Definido o tamanho do seu tabuleiro, basta gerar um vetor aleatório garantindo as restrições da representação da solução, ou seja:

- a) O vetor deve ser de tamanho  $n$  (quantidade de rainhas).
- b) Rainhas não ocupam a mesma linha e a mesma coluna nunca. Em outras palavras, o valores inteiros que representam as colunas onde as rainhas estão no vetor nunca serão repetidos.

**Exemplo de solução inicial válida para um tabuleiro com 8 rainhas**





# Busca Tabu

## Passo 2 – Vizinha da solução inicial

2. Avalie elementos na vizinhança da solução atual levando em conta movimentos que não sejam tabu ou que atendam a algum critério de aspiração.

Solução vizinha é aquela que possui uma ligeira perturbação/modificação em relação a solução original, por exemplo, se trocarmos as posições 3 e 8 neste vetor:

7	8	2	3	5	1	6	4
---	---	---	---	---	---	---	---

Teremos então uma solução vizinha a anterior que corresponderá a:

7	8	4	3	5	1	6	2
---	---	---	---	---	---	---	---

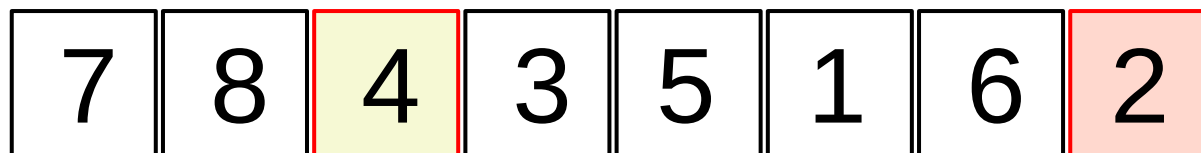
# Busca Tabu

## Passo 2 – Lista tabu

2. Avalie elementos na vizinhança da solução atual levando em conta movimentos que não sejam tabu ou que atendam a algum critério de aspiração.

As posições devem ser aleatoriamente selecionadas para serem trocadas. Após a troca ser realizada, você deverá armazenar o movimento realizado. Esse movimento realizado será chamado de movimento tabu, pois não poderá ser repetido até que se passem algumas iterações.

O controle destes movimentos é feito com a chamada lista tabu (matriz T).



# Busca Tabu

## Lista tabu – Memória de curto prazo

Armazena-se na parte superior da matriz  $T$ , na posição  $T_{ij}$ , por quanto tempo o movimento realizado ficará congelado (em qual iteração ele pode voltar a ser repetido).

Isso é chamado de **memória de curto prazo** no contexto do algoritmo.

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
5								
6								
7								
8								

# Busca Tabu

## Lista tabu – Memória de curto prazo

Por exemplo: Estou na iteração 1 e realizei uma mudança nos valores entre as posições 3 e 8 do vetor. O algoritmo agora deverá ser configurado para manter o movimento entre essas posições como proibido (tabu) por 3 iterações. Logo, só poderá voltar a ser feito na iteração 4, como mostra a lista tabu.

	1	2	3	4	5	6	7	8
1								
2								
3								4
4								
5								
6								
7								
8								

# Busca Tabu

## Lista tabu – Memória de longo prazo

Na parte inferior da matriz  $T$ , na posição  $T_{ji}$ , guardamos o número de vezes em que um movimento foi realizado.

Isso é chamado de **memória de longo prazo** no contexto do algoritmo.

Ou seja, como trocamos as posições 3 e 8 uma vez até aqui, então contabilizamos isso.

	1	2	3	4	5	6	7	8
1								
2								
3								4
4								
5								
6								
7								
8			1					

# Busca Tabu

## Passo 2 – Critério de aspiração

2. Avalie elementos na vizinhança da solução atual levando em conta movimentos que não sejam tabu ou que atendam a algum critério de aspiração.

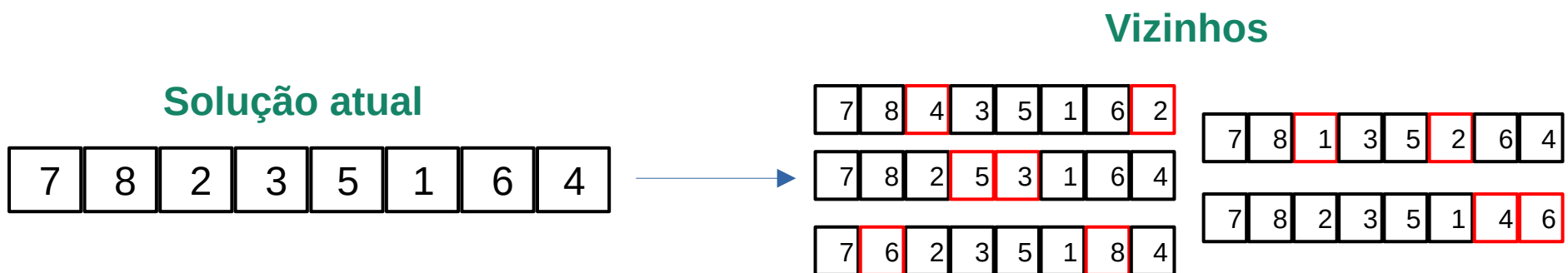
Considerando então que uma lista tabu existe, certos vetores não poderão ser criados em certos momentos. No entanto, isso pode ser burlado por algum critério de aspiração. O critério de aspiração é alguma regra que pode ser criada para que, mesmo que um movimento esteja proibido na lista tabu, ele seja realizado mesmo assim. Pode ser útil quando todos os movimentos ficam proibidos. Nesse caso, pode-se optar, por exemplo, por fazer o movimento que menos aconteceu.

# Busca Tabu

## Passo 2 – Geração de vizinhos

2. Avalie elementos na vizinhança da solução atual levando em conta movimentos que não sejam tabu ou que atendam a algum critério de aspiração.

Todos os vizinhos gerados devem então serem avaliados segundo a sua função objetivo. No caso, para verificar se alguma das novas soluções criadas apresentam uma quantidade menor de conflito entre rainhas do que a solução inicial apresentava.



# Busca Tabu

## Passo 3 – Seleção do melhor vizinho

3. Tome a **melhor solução vizinha** como a **solução atual**

Você deve ir gerando vizinhos até encontrar um que seja melhor do que a solução atual. Quando isso acontecer, esse melhor vizinho deve substituir a solução atual como nova melhor solução.

**Solução atual anterior**

7	8	2	3	5	1	6	4
---	---	---	---	---	---	---	---



**Solução atual** ← **Melhor vizinho da antiga solução atual**

7	8	4	3	5	1	6	2
---	---	---	---	---	---	---	---



# Busca Tabu

## Passo 4 – Atualização da lista tabu

### 4. Atualize a *lista tabu*

A cada tentativa de gerar uma melhor solução pela modificação de posições na solução atual (processo de geração de vizinhos) você deve sempre ir atualizando a lista tabu com o movimento que você realizou.

Você perceberá conferindo a lista tabu em pontos avançados do algoritmo que certos movimentos tem maior tendência de acontecer, pois é onde estão as rainhas nas posições corretas e que geram menos conflitos entre elas.

	1	2	3	4	5	6	7	8
1					25		22	
2	2					22		
3						25		21
4	2							
5	1	5		1				
6	1		15					
7	1	3		1				
8		3	10					

# Busca Tabu

## Passo 5 – Critérios de parada

5. Se o **critério de parada** não for satisfeito, volte ao passo 2, senão termine o algoritmo.

O sua Busca Tabu em um tabuleiro muito grande poderia ficar executando por horas, dias, meses.... é necessário existir algum critério de parada.

O critério de parada mais óbvio é parar o algoritmo quando uma solução ótimo seja encontrada. Em outras palavras, uma solução com custo zero, ou seja, que não apresente nenhum conflito entre rainhas.

Outro critério é definir uma quantidade máxima de iterações que serão realizadas no algoritmo, visto que se isso não for feito ele poderá ficar executando para sempre.

**Vamos conferir  
uma  
implementação  
do Busca Tabu  
para o  
N-Rainhas no **R**!**

