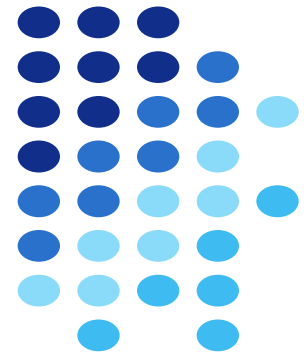


Universidade Federal de Sergipe
Departamento de Sistemas de Informação
SINF0007 – Estrutura de Dados II

Backtracking

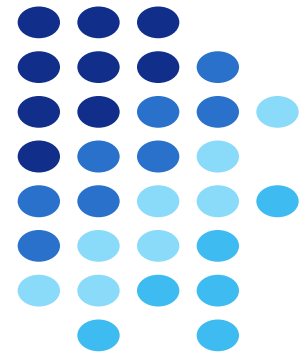


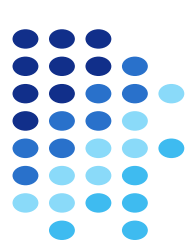
9

Prof. Dr. Raphael Pereira de Oliveira

raphael.oliveira@academico.ufs.br

Introdução

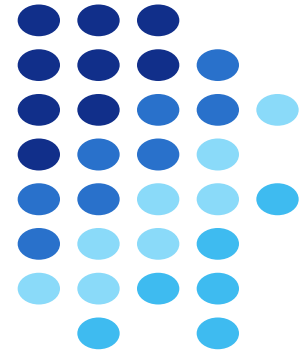


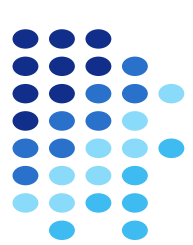


Introdução

- Dado um problema computacional, podemos resolvê-lo usando a **força bruta**
 - **ex.:** muitas vezes é mais fácil contar o número de vezes que um item aparece no *array* do que encontrar uma forma matemática
 - Ou seja, testar todas as soluções candidatas
- Em geral isso é pouco eficiente, pois: Verificar se uma solução é viável pode ser custoso
- O número de soluções candidatas costuma ser muito grande

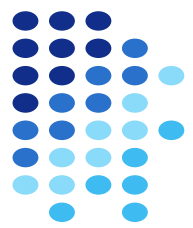
Backtracking





Backtracking

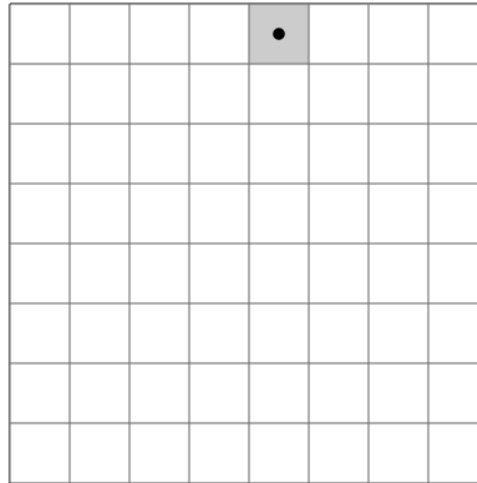
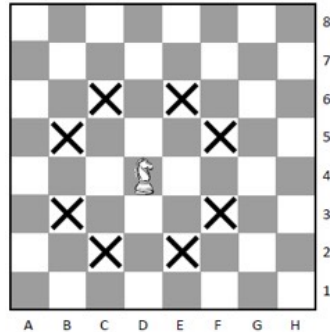
- Método sistemático para iterar em todas as possíveis configurações de um espaço de busca
- Precisa ser customizado para cada problema específico
- Ideia central é retroceder quando detectar que a solução candidata é inviável

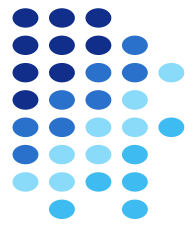


Backtracking – Exemplo

Problema: **Passeio do Cavalo**

- Em um tabuleiro $n \times n$, partindo da posição dada, encontrar, se existir, um passeio do cavalo que visita os todos os pontos do tabuleiro uma única vez

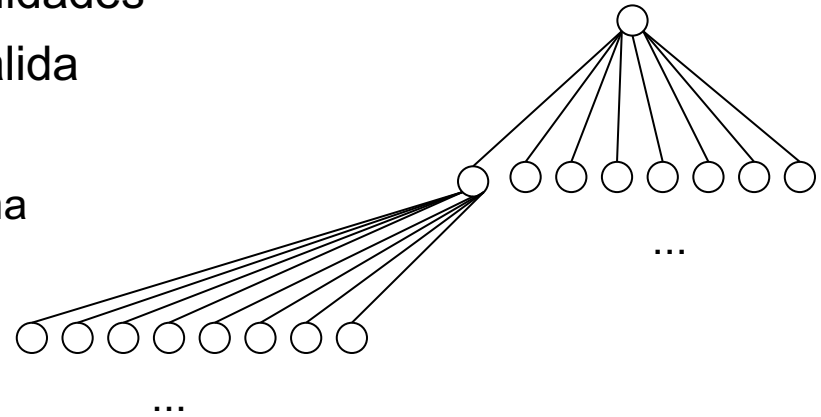


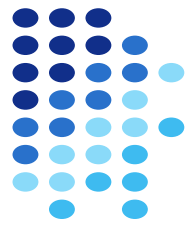


Backtracking – Exemplo: Possível Estratégia

Problema: **Passeio do Cavalo**

- A partir da primeira posição: 8 possibilidades
- Identifica as válidas e Escolhe uma válida
- A partir da escolhida
 - Identifica as válidas e Escolhe mais uma
 - ... e assim por diante
- Deu errado?
 - Volta e tenta outro caminho (backtracking)
- Cada nó é uma casa do tabuleiro (Se o tabuleiro for 8x8, quando atingirmos o nó 64, encontramos uma solução)



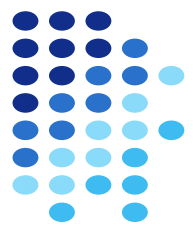


Backtracking – Representação

- De forma geral, vemos a nossa solução como um ***array***

$$a = (a_1, a_2, a_3 \dots a_n)$$

- Esse *array* pode significar muitas coisas, dependendo do problema:
 - Cada a_i pode significar se o elemento está presente ou não na solução
 - A configuração de cada coluna do tabuleiro
 - A sequência de operações de uma pilha



Backtracking – Algoritmo

- A cada passo do algoritmo, começamos com uma solução parcial, ex.: $\mathbf{a} = (a_1, a_2, a_3 \dots a_k)$ e tentamos aumentar adicionando outro elemento no final da solução
- Depois de adicionar, devemos testar se já chegamos na solução procurada
- Se sim, imprima, conte, armazene, faça o que o problema pede
- Se não, temos que verificar se ainda é possível outra alternativa para a solução parcial chegar a solução completa
- Se sim, continua
- Se não, remova o último elemento de \mathbf{a} e tente outra possibilidade

Backtracking – Algoritmo

```
bool finished = false; // Já encontrou todas as soluções?
```

```
void backtrack(int a[], int k, data input){
```

```
    int c[MAXCANDIDATES]; // Candidatos para a próxima posição
```

```
    int nCandidates; // total de candidatos para próxima posição
```

```
    if(is_a_solution(a, k, input))
```

```
        process_Solution(a, k, input); // Se for uma solução, imprima, guarde, ...
```

```
    else{
```

```
        k = k + 1; // Se ainda não chegou na solução,
```

```
            //vamos tentar com mais um passo (desce na árvore, anda no array, ...)
```

```
        construct_candidates(a, k, input, c, &nCandidates); // Tenta somente nos candidatos
```

```
                                // válidos (olhar restrições do problema)
```

```
        for (int i=0; i < nCandidates; i++){
```

```
            a[k] = c[i]; // considera que c[i] é uma possível solução
```

```
            backtrack(a, k, input);
```

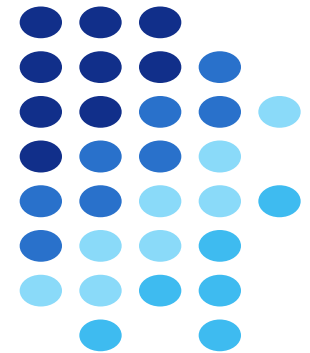
```
            if (finished) return; // término antecipado
```

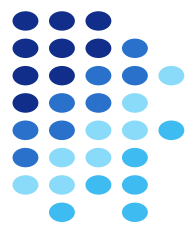
```
        }
```

```
    }
```

```
}
```

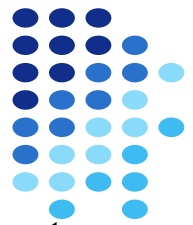
Exemplo Gerando Sequências





Exemplo - Gerando Sequências

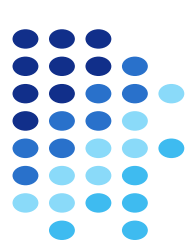
- Como imprimir todas as sequências de tamanho **k** de números entre **1** e **n**?
- Exemplo: **n** = 4, **k** = 3



Exemplo - Gerando Sequências

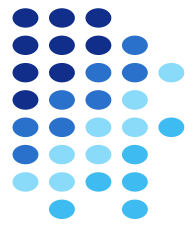
- Como imprimir todas as sequências de tamanho **k** de números entre **1** e **n**?
- Exemplo: **n = 4**, **k = 3**

1 1 1	1 3 1	2 1 1	2 3 1	3 1 1	3 3 1	4 1 1	4 3 1
1 1 2	1 3 2	2 1 2	2 3 2	3 1 2	3 3 2	4 1 2	4 3 2
1 1 3	1 3 3	2 1 3	2 3 3	3 1 3	3 3 3	4 1 3	4 3 3
1 1 4	1 3 4	2 1 4	2 3 4	3 1 4	3 3 4	4 1 4	4 3 4
1 2 1	1 4 1	2 2 1	2 4 1	3 2 1	3 4 1	4 2 1	4 4 1
1 2 2	1 4 2	2 2 2	2 4 2	3 2 2	3 4 2	4 2 2	4 4 2
1 2 3	1 4 3	2 2 3	2 4 3	3 2 3	3 4 3	4 2 3	4 4 3
1 2 4	1 4 4	2 2 4	2 4 4	3 2 4	3 4 4	4 2 4	4 4 4



Exemplo - Gerando Sequências

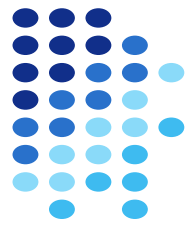
- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

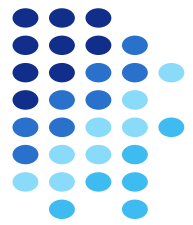




Exemplo - Gerando Sequências

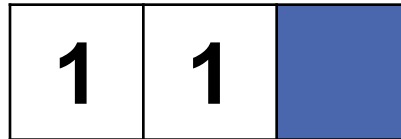
- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

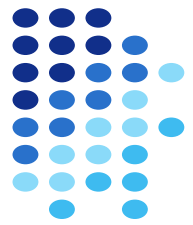




Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

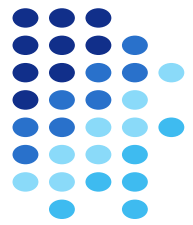




Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

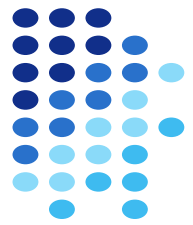
1	1	1
---	---	---



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

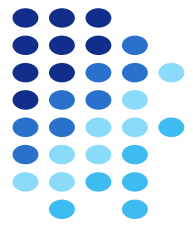
1	1	2
---	---	---



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

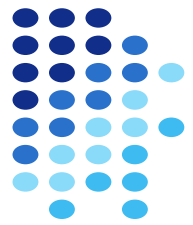
1	1	3
---	---	---



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

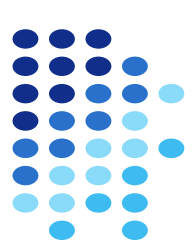
1	1	4
---	---	---



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

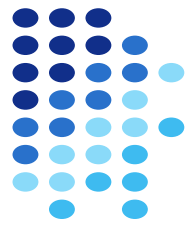
1	1	4
---	---	---



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

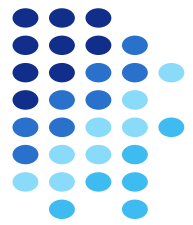
1	2	4
---	---	---



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

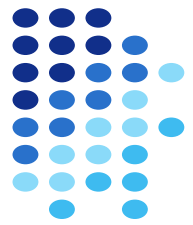
1	2	1
---	---	---



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

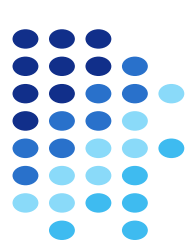
1	2	2
---	---	---



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

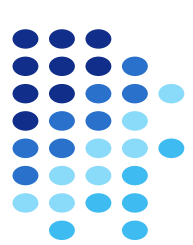
1	2	3
---	---	---



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

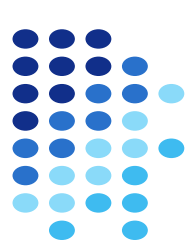
1	2	4
---	---	---



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

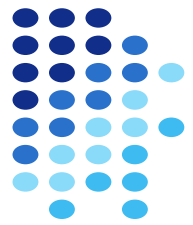
1	2	4
---	---	---



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

1	3	4
---	---	---



Exemplo - Gerando Sequências

- Podemos resolver usando **Recursão**:
 - Armazenamos o prefixo da sequência que estamos construindo
 - Completamos com todos os possíveis sufixos recursivamente
- Simulação para $n = 4$, $k = 3$:

1	3	1
---	---	---

...

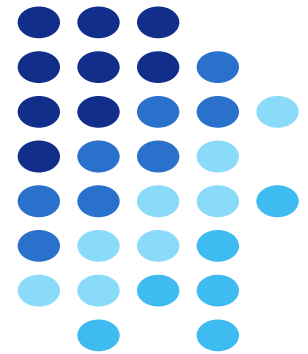
Exemplo - Gerando Sequências

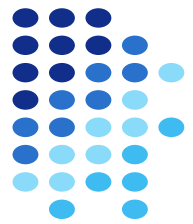
- Vamos tentar manter a lógica do algoritmo apresentado e implementar:
 - `Construct_candidates`
 - `is_a_solution`
 - `process_solution`

Ver código
sequence.c

Exemplo

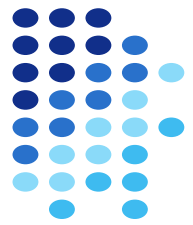
Gerando Subconjuntos





Exemplo - Gerando Subconjuntos

- Gere todos os subconjuntos de um conjunto
 - **Entrada:** um inteiro n , indicando o maior inteiro do conjunto de inteiros de 1 a n
 - **Saída:** todos os subconjuntos, um por linha, cada número separado por um espaço em branco. Cada subconjunto deve ser delimitado pelo caracteres $\{\}$. A saída deve ser ordenada por ordem lexicográfica.



Exemplo - Gerando Subconjuntos

Entrada

3

Saída

{ 1 2 3 }

{ 1 2 }

{ 1 3 }

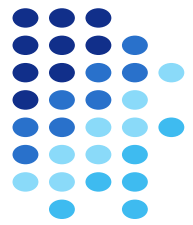
{ 1 }

{ 2 3 }

{ 2 }

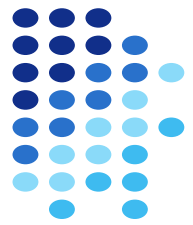
{ 3 }

{ }



Exemplo - Gerando Subconjuntos

- Vamos tentar manter a lógica do algoritmo apresentado e implementar:
 - `Construct_candidates`
 - `is_a_solution`
 - `process_solution`



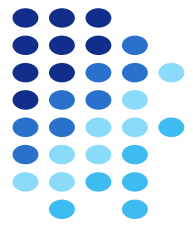
Exemplo - Gerando Subconjuntos

- Uma ideia é representar a solução como um *array*
 - Para ilustrar o uso da árvore como forma de pensar no problema

Exemplo: se $n = 3$, podemos representar os conjuntos

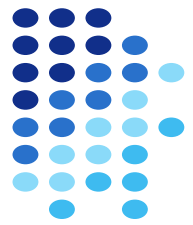
$\{ 1 \ 2 \ 3 \}$, $\{ 1 \ 2 \}$, $\{ 1 \ 3 \}$ e $\{ 2 \}$ pelos *arrays*:

- $[0,1,1,1]$
- $[0,1,1,0]$
- $[0,1,0,1]$
- $[0,0,1,0]$



Exemplo - Gerando Subconjuntos

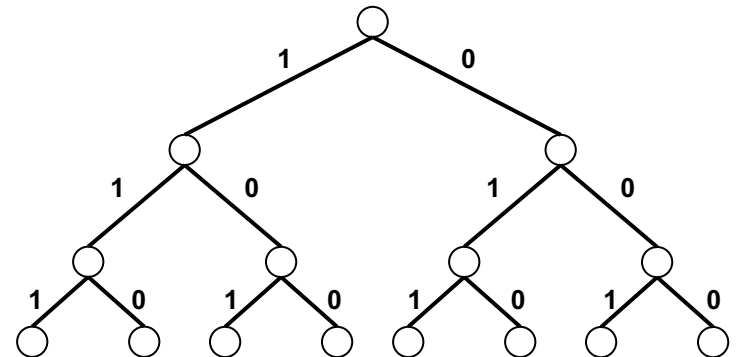
- Dessa forma, cada posição i do array, representa se o número i está ou não presente no conjunto
- Para soluções com ***backtracking*** o segredo é pensar em representar o problema de forma a construir a solução a cada passo

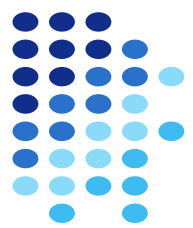


Exemplo - Gerando Subconjuntos

Representado o conjunto { 1 3 }

- No 1º momento, teremos 2 opções (candidatos), o **1** fará (1) ou não (0) parte do conjunto? *Suponha que sim (1).*





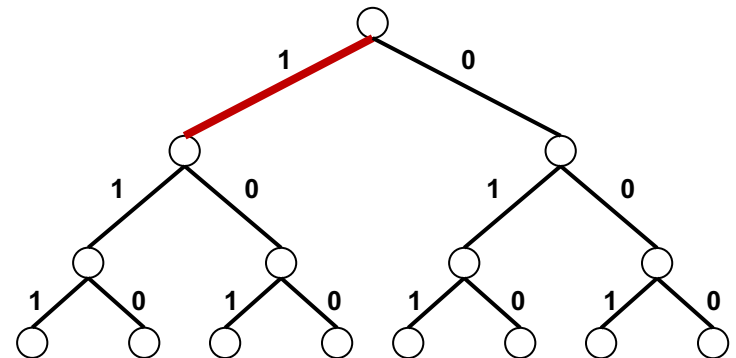
Exemplo - Gerando Subconjuntos

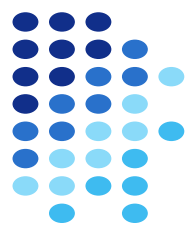
Representado o conjunto { 1 3 }

- No 1º momento, teremos 2 opções (candidatos), o **1** fará (1) ou não (0) parte do conjunto? *Suponha que sim (1).*

- Array de solução

(índices do array)	0	1	2	3
		1		





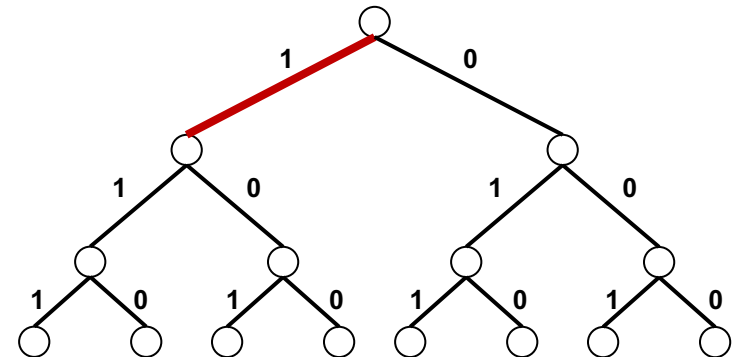
Exemplo - Gerando Subconjuntos

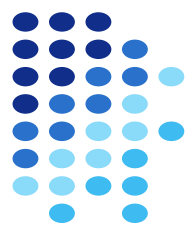
Representado o conjunto { 1 3 }

- No 1º momento, teremos 2 opções (candidatos), o **1** fará (1) ou não (0) parte do conjunto? *Suponha que sim (1).*
- Então, descemos na árvore. De novo, teríamos a mesma opção, o **2** fará ou não parte do conjunto? *Suponha que não (0).*

- Array de solução

(índices do array)	0	1	2	3
		1		





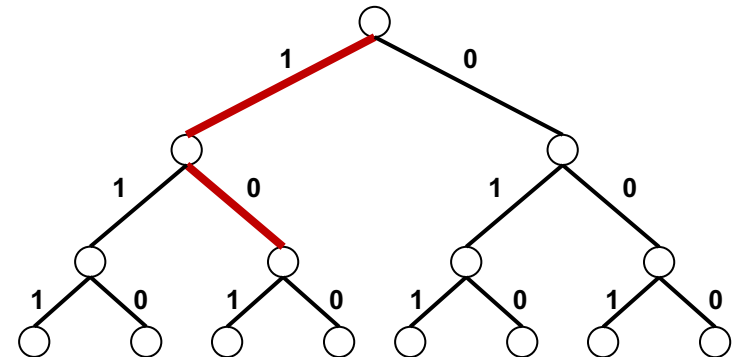
Exemplo - Gerando Subconjuntos

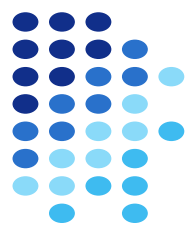
Representado o conjunto { 1 3 }

- No 1º momento, teremos 2 opções (candidatos), o **1** fará (1) ou não (0) parte do conjunto? *Suponha que sim (1).*
- Então, descemos na árvore. De novo, teríamos a mesma opção, o **2** fará ou não parte do conjunto? *Suponha que não (0).*

- Array de solução

(índices do array)	0	1	2	3
		1	0	





Exemplo - Gerando Subconjuntos

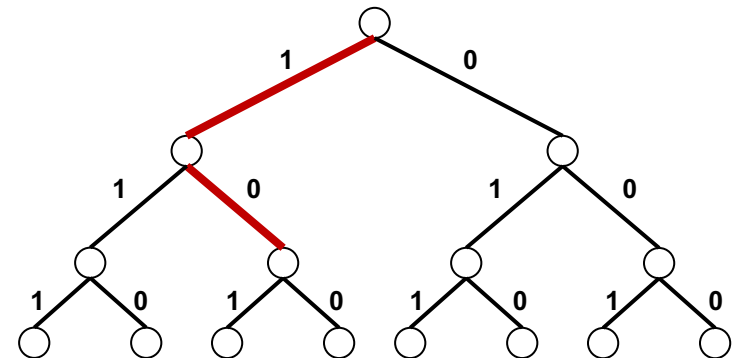
Representado o conjunto { 1 3 }

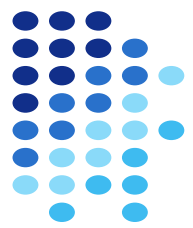
- No 1º momento, teremos 2 opções (candidatos), o **1** fará (1) ou não (0) parte do conjunto? *Suponha que sim (1).*
- Então, descemos na árvore. De novo, teríamos a mesma opção, o **2** fará ou não parte do conjunto? *Suponha que não (0).*
- Na próxima descida, o **3** fará parte ou não? *Suponha que sim (1).*

- Array de solução

(índices do array) 0 1 2 3

	1	0	





Exemplo - Gerando Subconjuntos

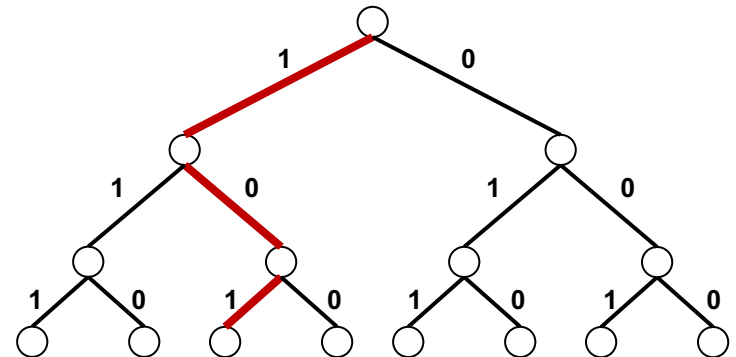
Representado o conjunto { 1 3 }

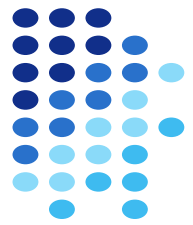
- No 1º momento, teremos 2 opções (candidatos), o **1** fará (1) ou não (0) parte do conjunto? *Suponha que sim (1).*
- Então, descemos na árvore. De novo, teríamos a mesma opção, o **2** fará ou não parte do conjunto? *Suponha que não (0).*
- Na próxima descida, o **3** fará parte ou não? *Suponha que sim (1).*

- Array de solução

(índices do array) 0 1 2 3

	1	0	1





Exemplo - Gerando Subconjuntos

Representado o conjunto { 1 3 }

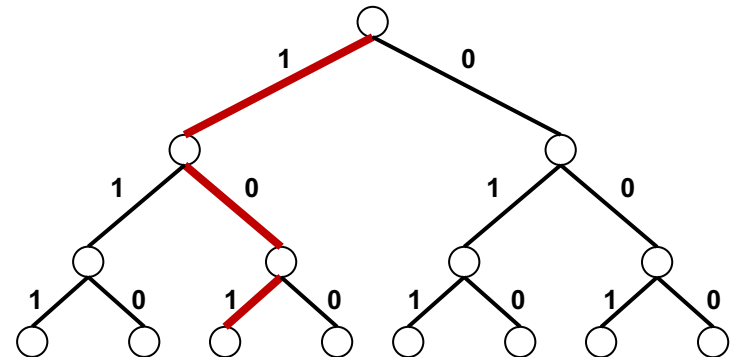
- No 1º momento, teremos 2 opções (candidatos), o **1** fará (1) ou não (0) parte do conjunto? *Suponha que sim (1).*
- Então, descemos na árvore. De novo, teríamos a mesma opção, o **2** fará ou não parte do conjunto? *Suponha que não (0).*
- Na próxima descida, o **3** fará parte ou não? *Suponha que sim (1).*

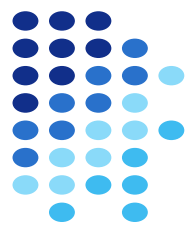
- Array de solução

(índices do array) 0 1 2 3

	1	0	1

Esse array representa o conjunto
{ 1 3 }



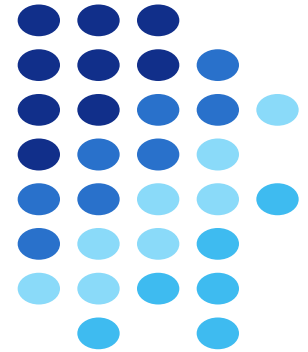


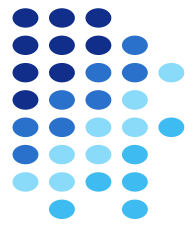
Exemplo - Gerando Subconjuntos

- Assim, tudo o que temos que fazer é andar na árvore e quando chegar nas folhas, imprimir o *array* de solução, que estará com 0's e 1's

Ver código
subSets.c

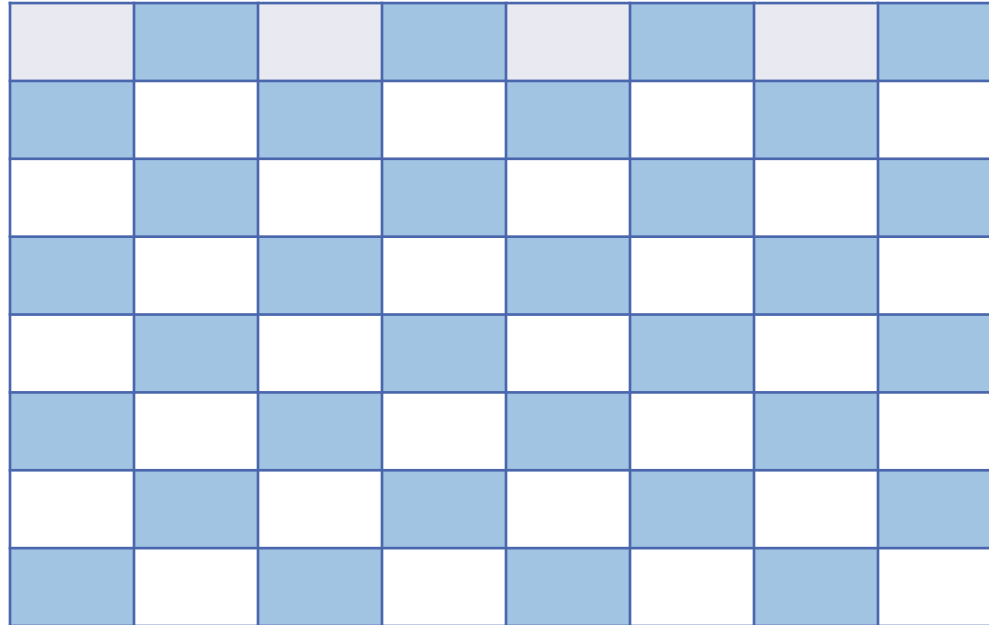
Exemplo 8 Rainhas

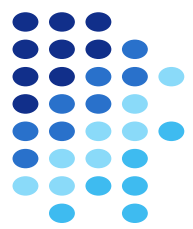




Exemplo – 8 Rainhas

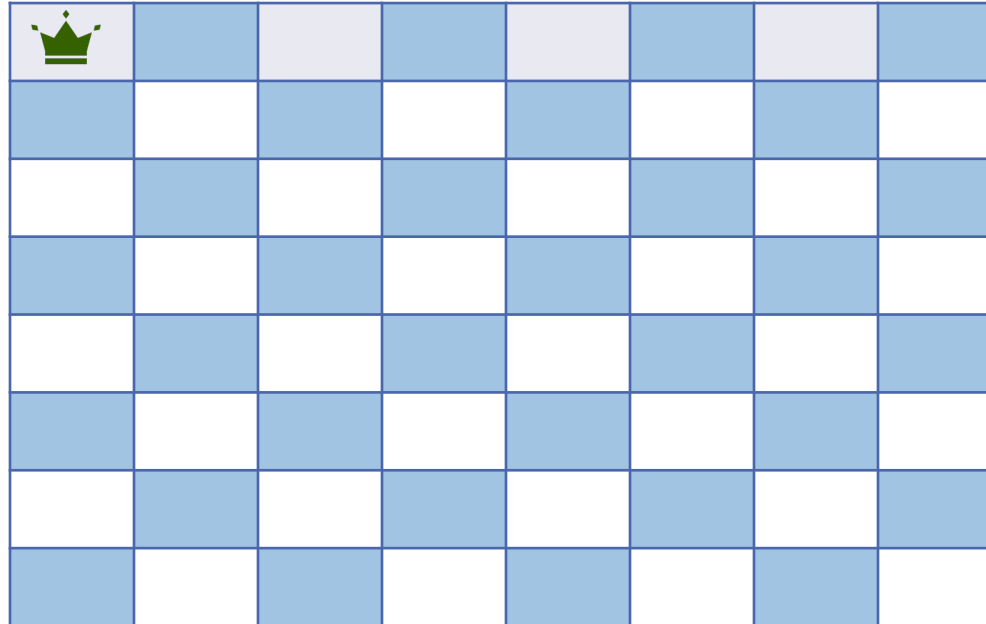
- No xadrez é possível colocar 8 rainhas no tabuleiro de tal forma que nenhuma rainha possa atacar a outra

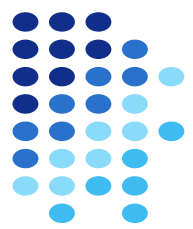




Exemplo – 8 Rainhas

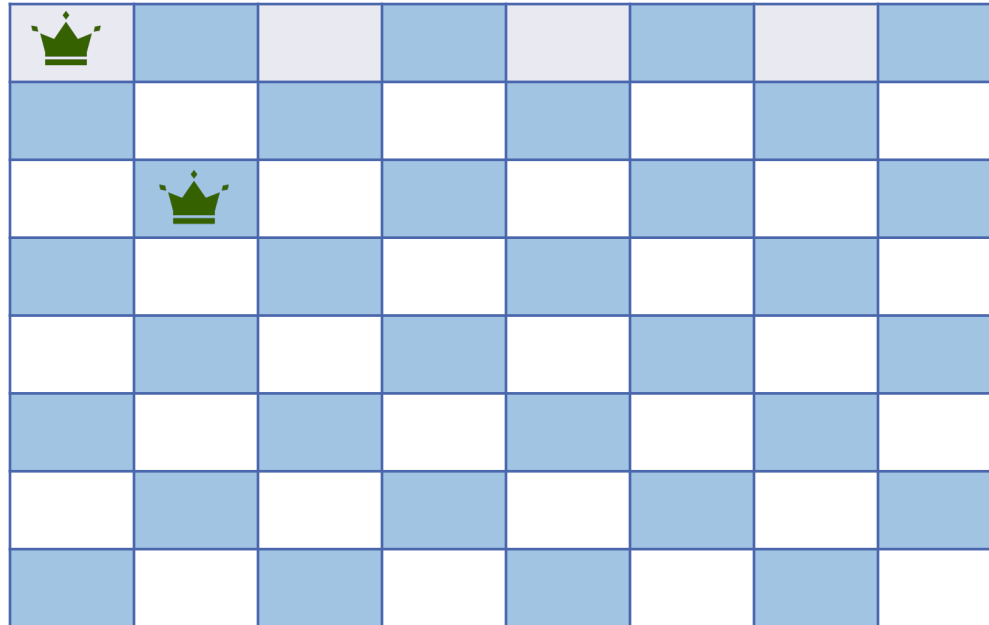
- No xadrez é possível colocar 8 rainhas no tabuleiro de tal forma que nenhuma rainha possa atacar a outra. **Ex.:**

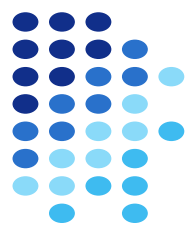




Exemplo – 8 Rainhas

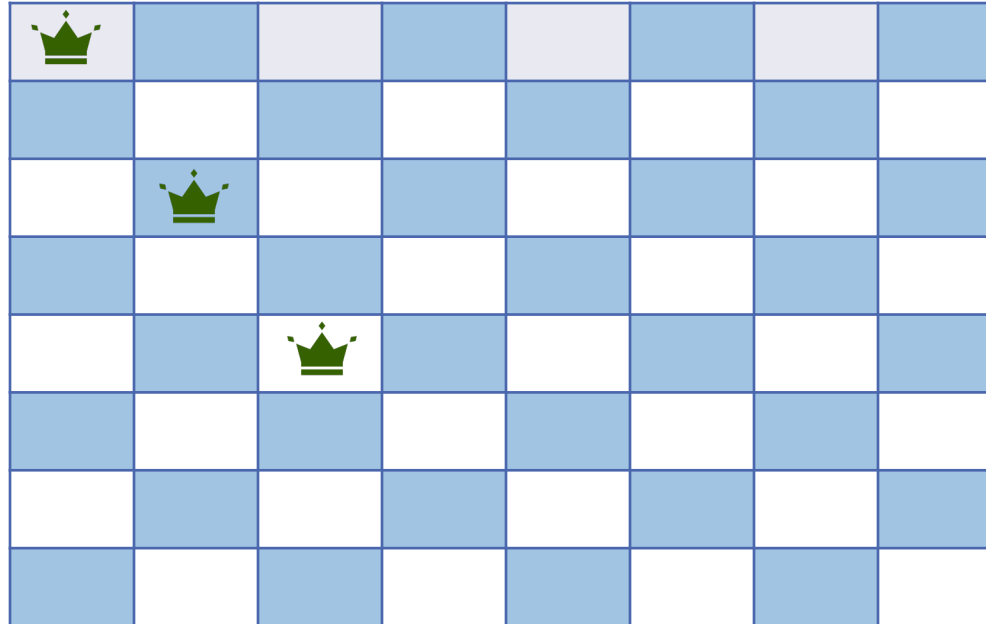
- No xadrez é possível colocar 8 rainhas no tabuleiro de tal forma que nenhuma rainha possa atacar a outra. **Ex.:**

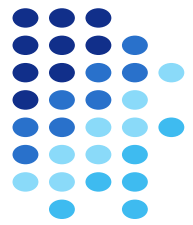




Exemplo – 8 Rainhas

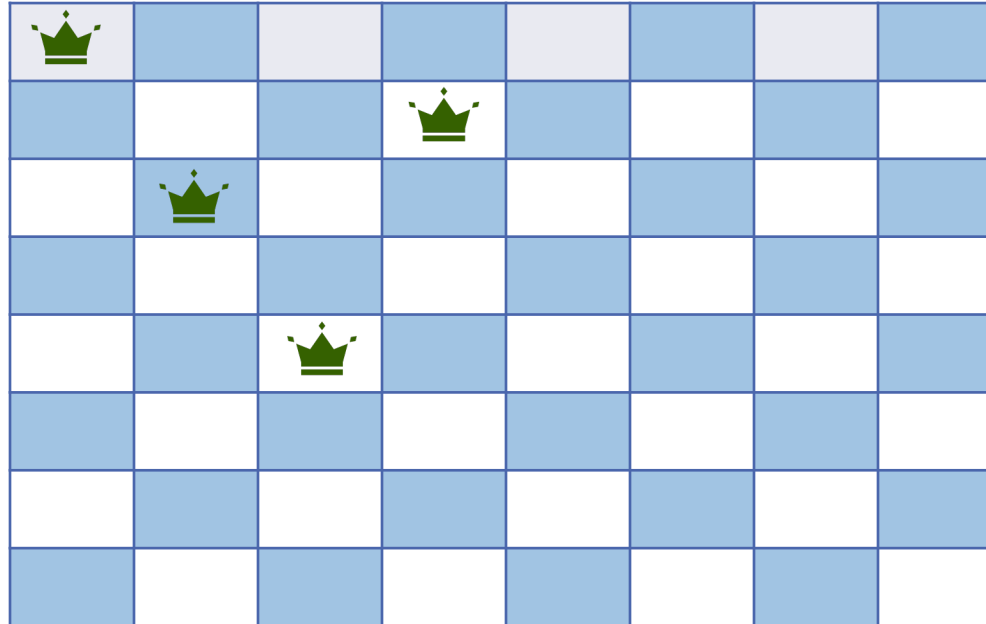
- No xadrez é possível colocar 8 rainhas no tabuleiro de tal forma que nenhuma rainha possa atacar a outra. **Ex.:**

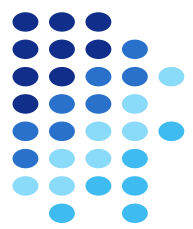




Exemplo – 8 Rainhas

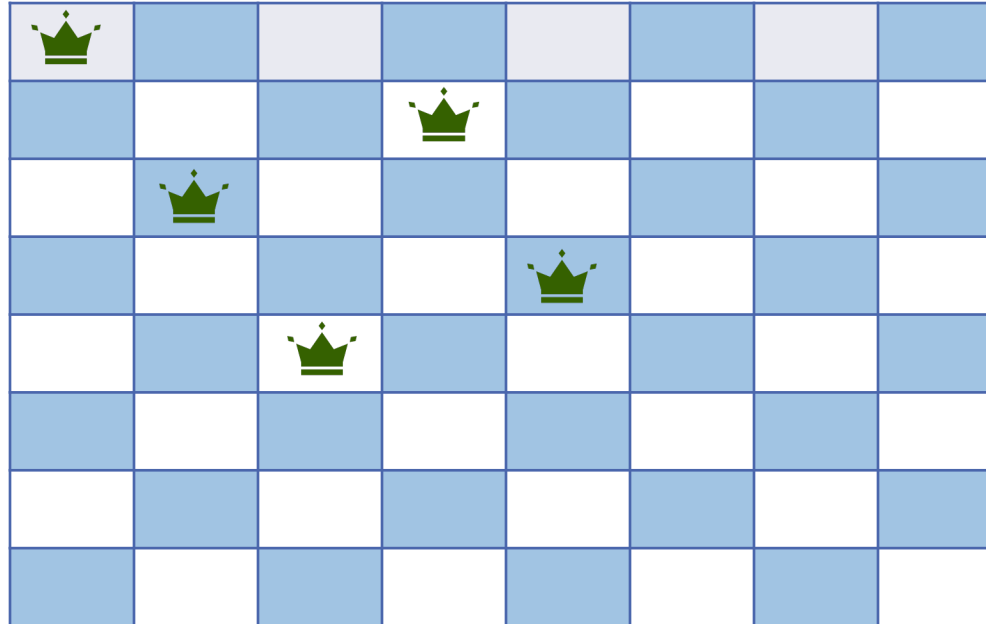
- No xadrez é possível colocar 8 rainhas no tabuleiro de tal forma que nenhuma rainha possa atacar a outra. **Ex.:**

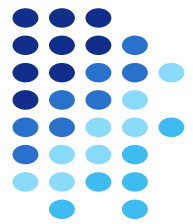




Exemplo – 8 Rainhas

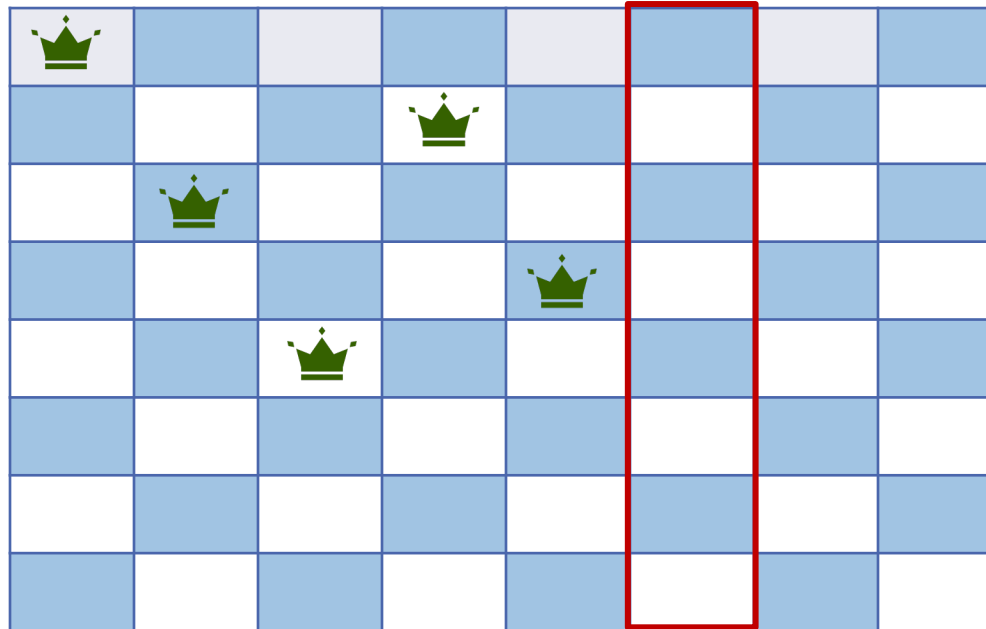
- No xadrez é possível colocar 8 rainhas no tabuleiro de tal forma que nenhuma rainha possa atacar a outra. **Ex.:**

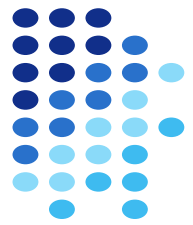




Exemplo – 8 Rainhas

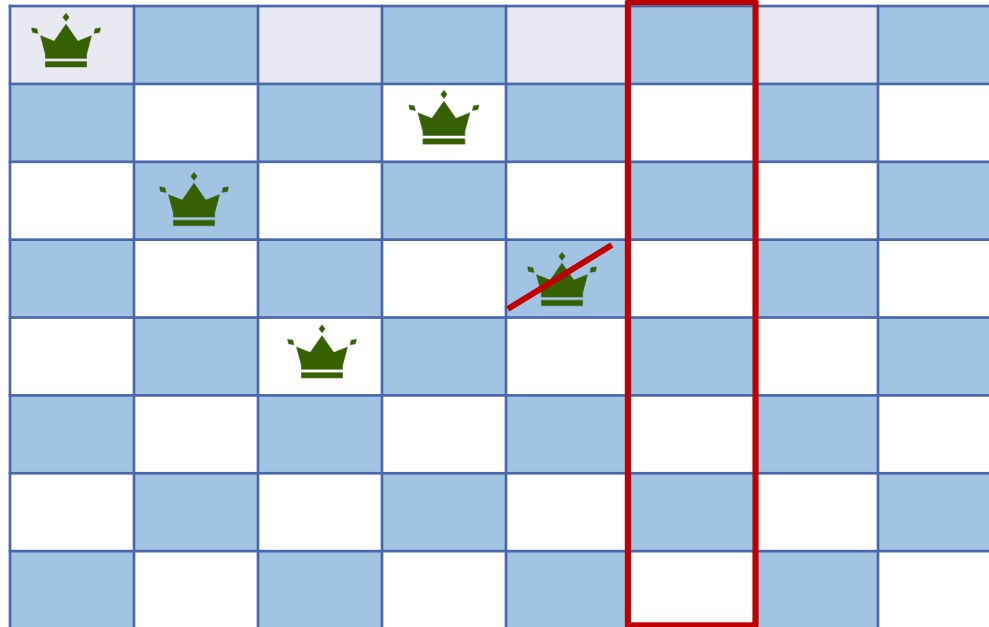
- No xadrez é possível colocar 8 rainhas no tabuleiro de tal forma que nenhuma rainha possa **?** ar a outra. **Ex.:**



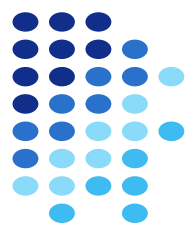


Exemplo – 8 Rainhas

- No xadrez é possível colocar 8 rainhas no tabuleiro de tal forma que nenhuma rainha possa atacar a outra. **Ex.:**

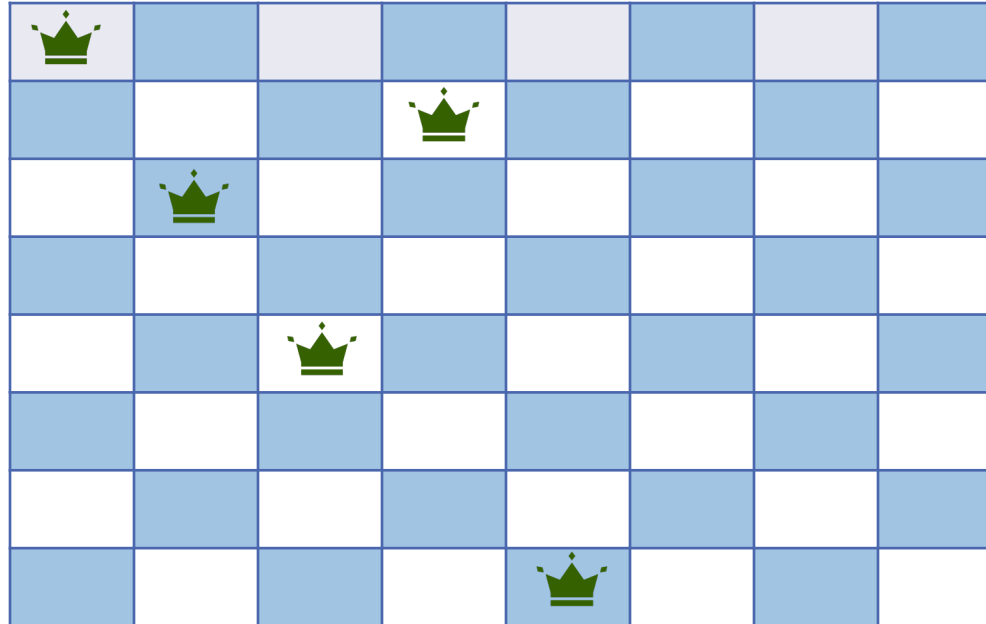


0 candidatos!
Temos que fazer o
backtracking e
mudar o candidato
anterior!

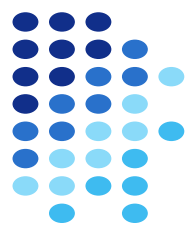


Exemplo – 8 Rainhas

- No xadrez é possível colocar 8 rainhas no tabuleiro de tal forma que nenhuma rainha possa atacar a outra. **Ex.:**



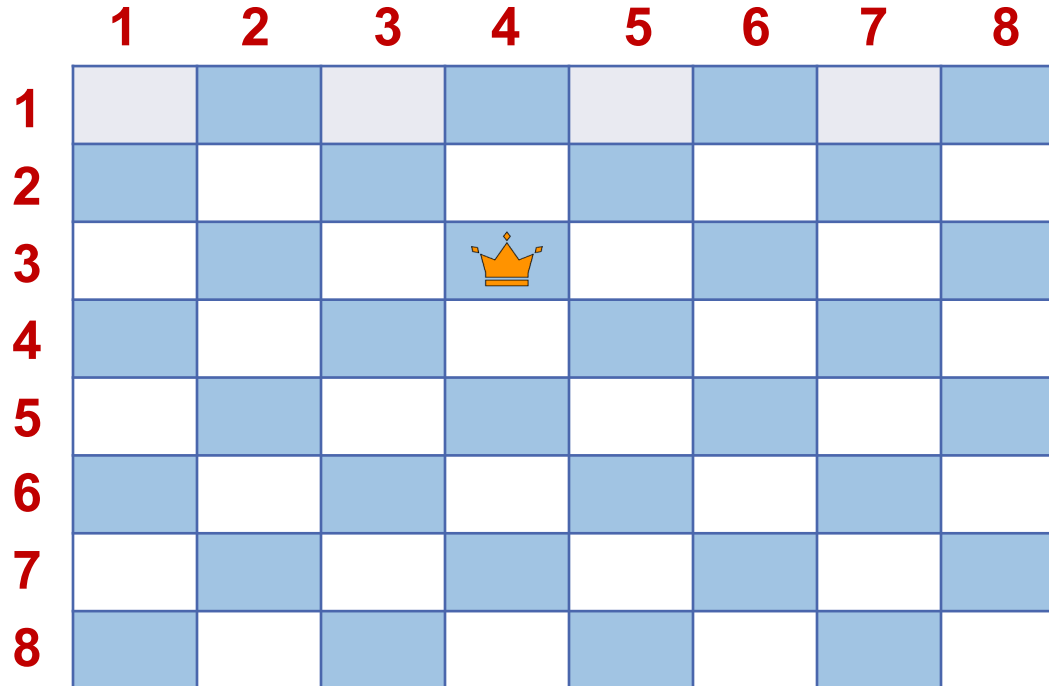
... E assim
sucessivamente!

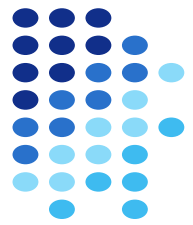


Exemplo – 8 Rainhas

Contudo...

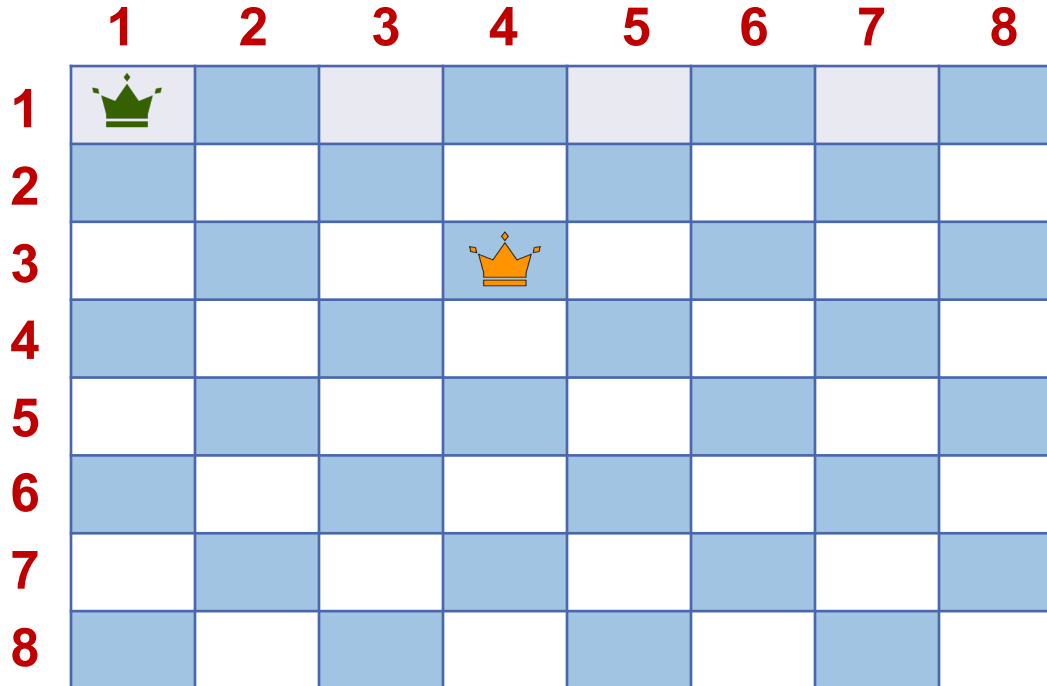
- O problema já define o local de **1 rainha FIXA**

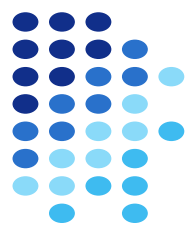




Exemplo – 8 Rainhas

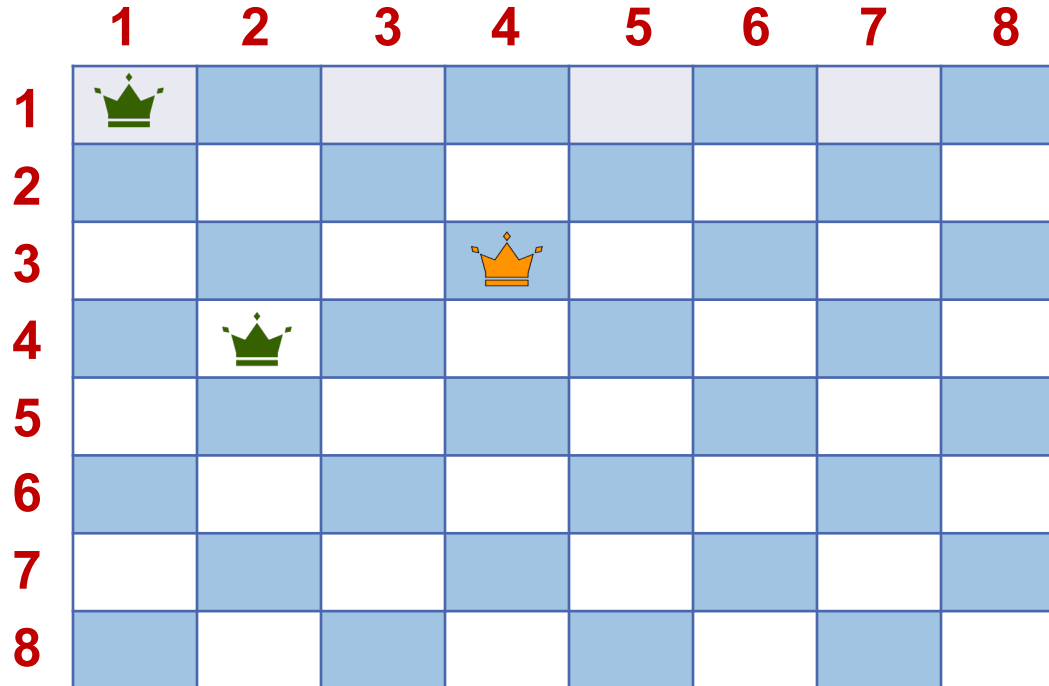
- O problema já define o local de **1 rainha FIXA**. Ex.:

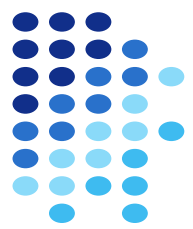




Exemplo – 8 Rainhas

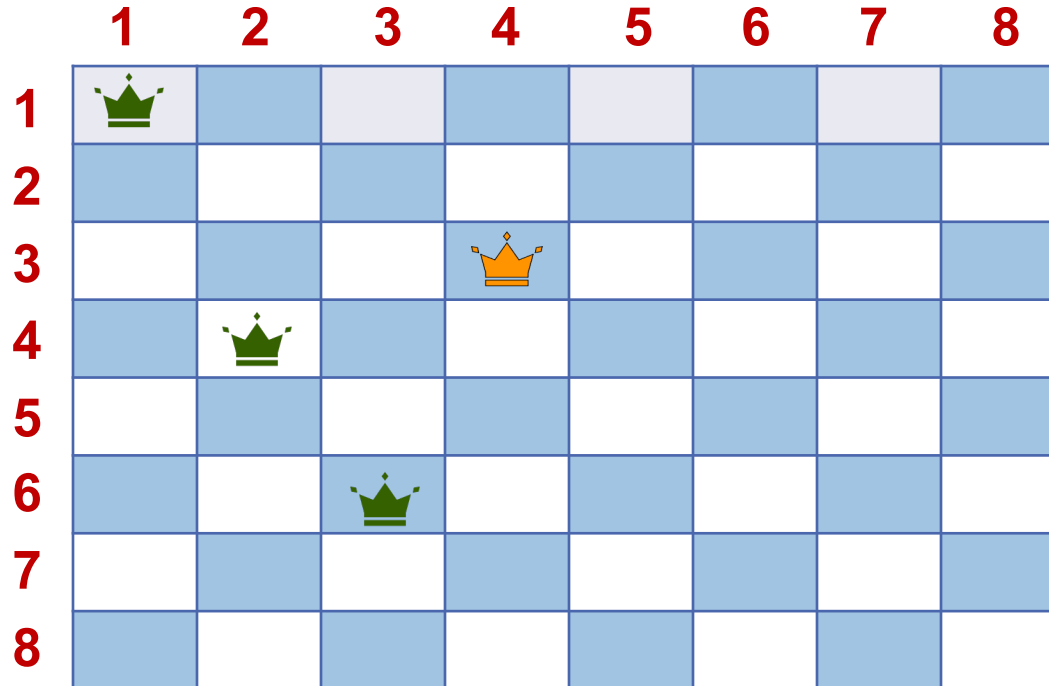
- O problema já define o local de **1 rainha FIXA**. Ex.:

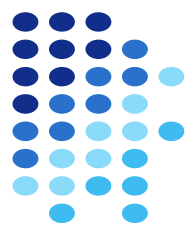




Exemplo – 8 Rainhas

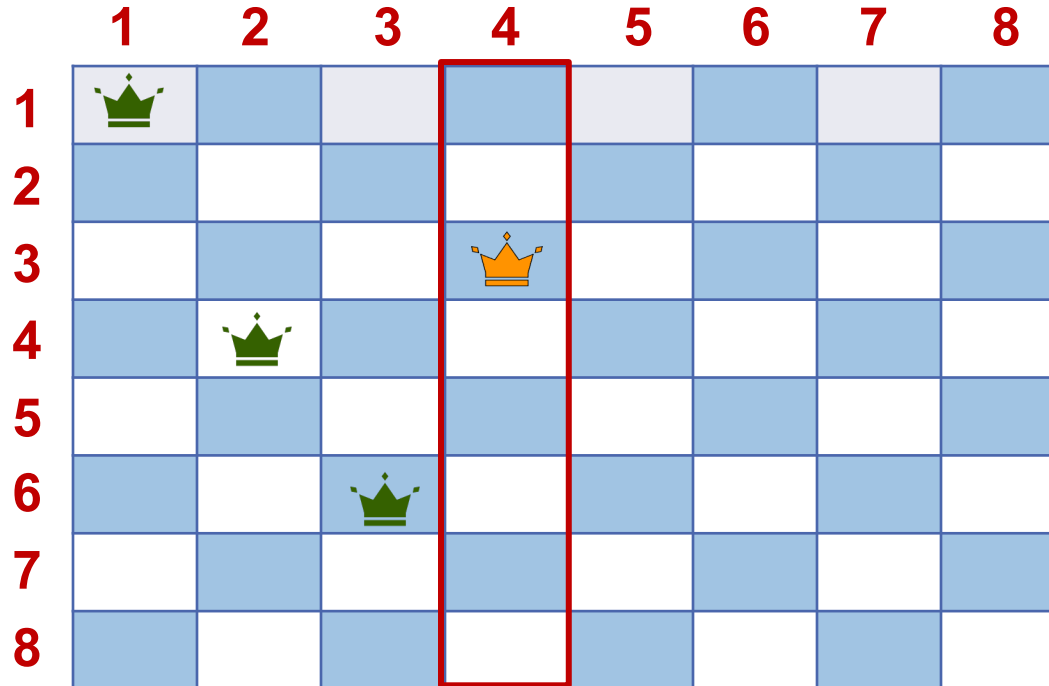
- O problema já define o local de **1 rainha FIXA**. Ex.:



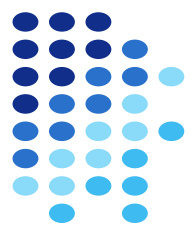


Exemplo – 8 Rainhas

- O problema já define o local de **1 rainha FIXA**. Ex.:

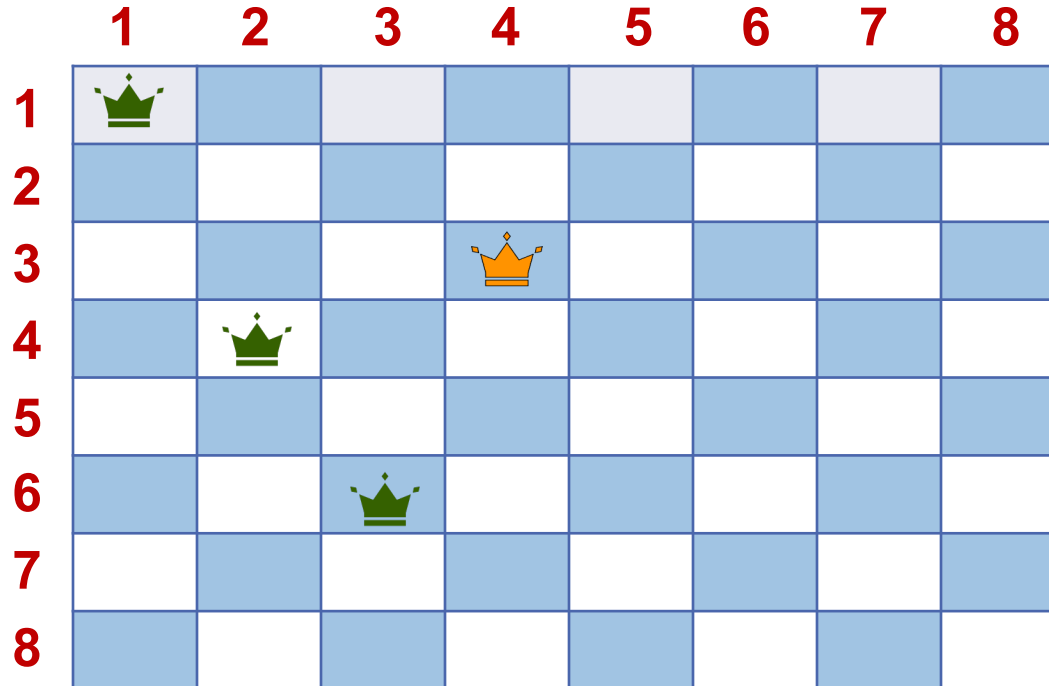


É a rainha **FIXA**,
temos que pular!

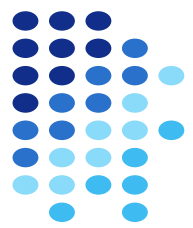


Exemplo – 8 Rainhas

- O problema já define o local de **1 rainha FIXA**. Ex.:

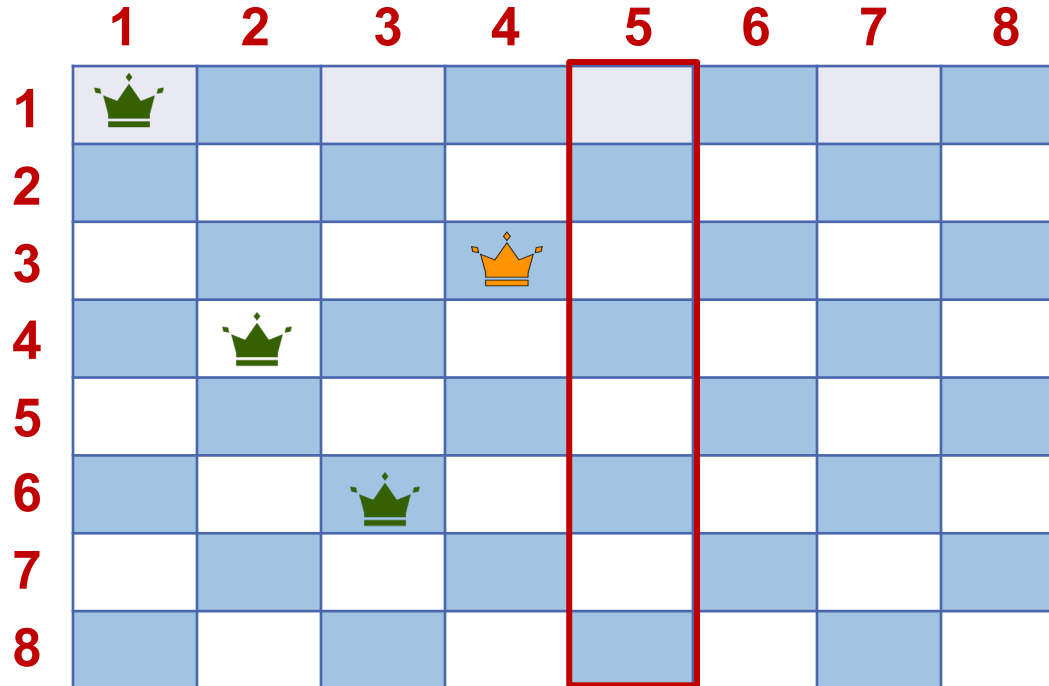


0 candidatos!



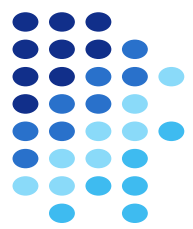
Exemplo – 8 Rainhas

- O problema já define o local de **1 rainha FIXA**. Ex.:



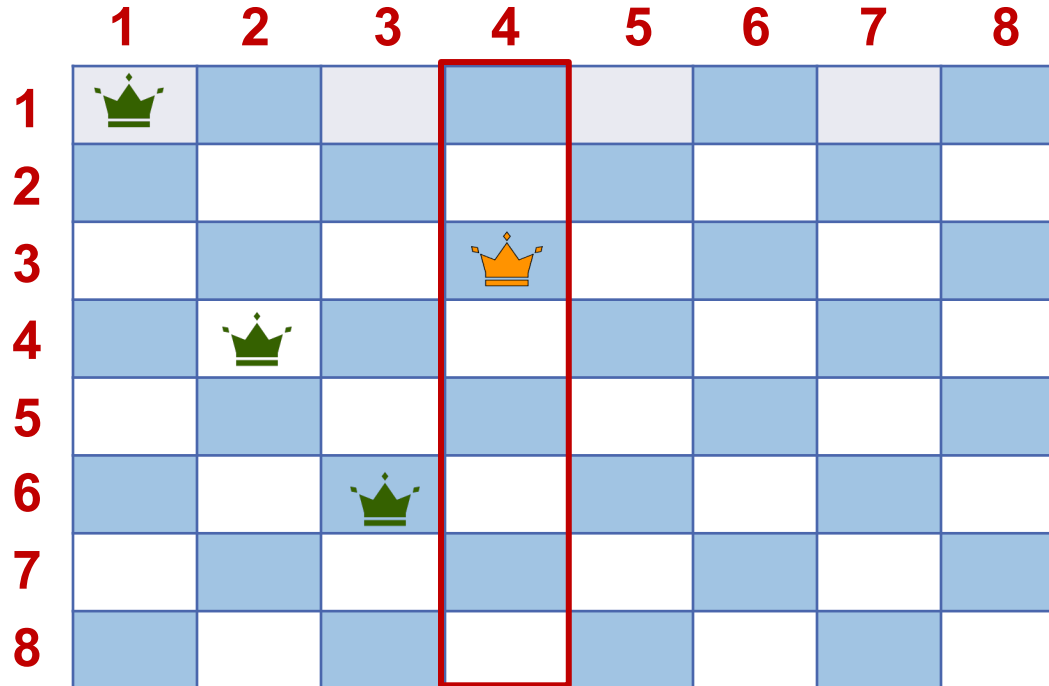
0 candidatos!

Temos que fazer o
backtracking e
mudar o candidato
anterior!

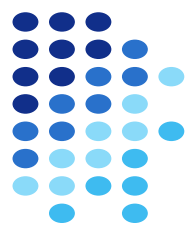


Exemplo – 8 Rainhas

- O problema já define o local de **1 rainha FIXA**. Ex.:

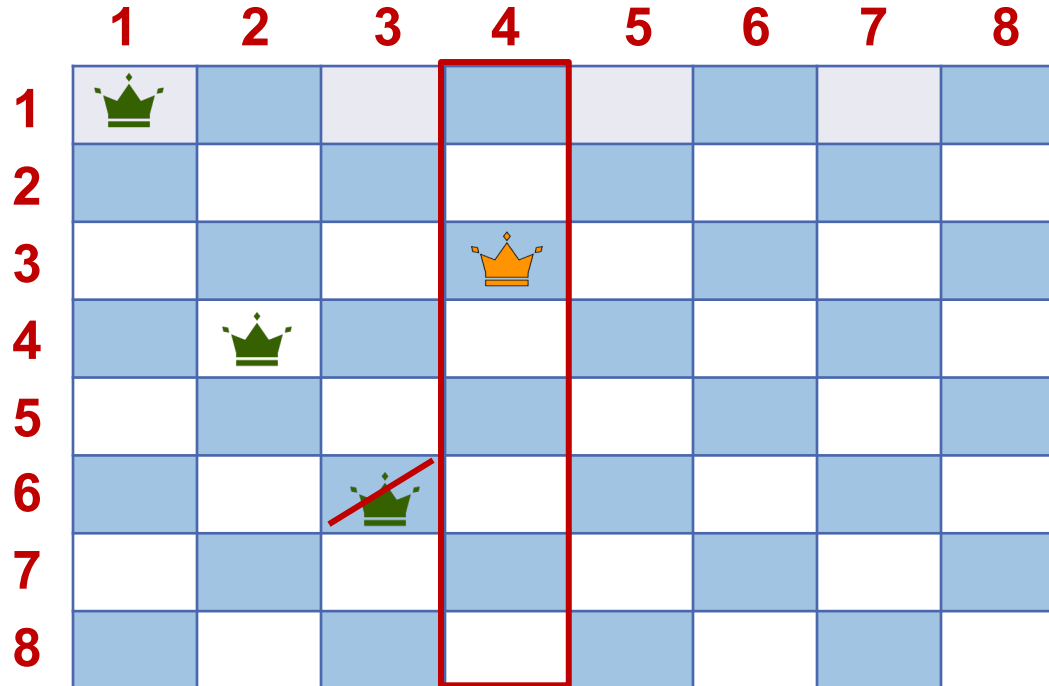


O candidato anterior é a rainha FIXA! Fazemos o **backtracking** mais uma vez!

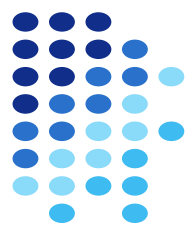


Exemplo – 8 Rainhas

- O problema já define o local de **1 rainha FIXA**. Ex.:

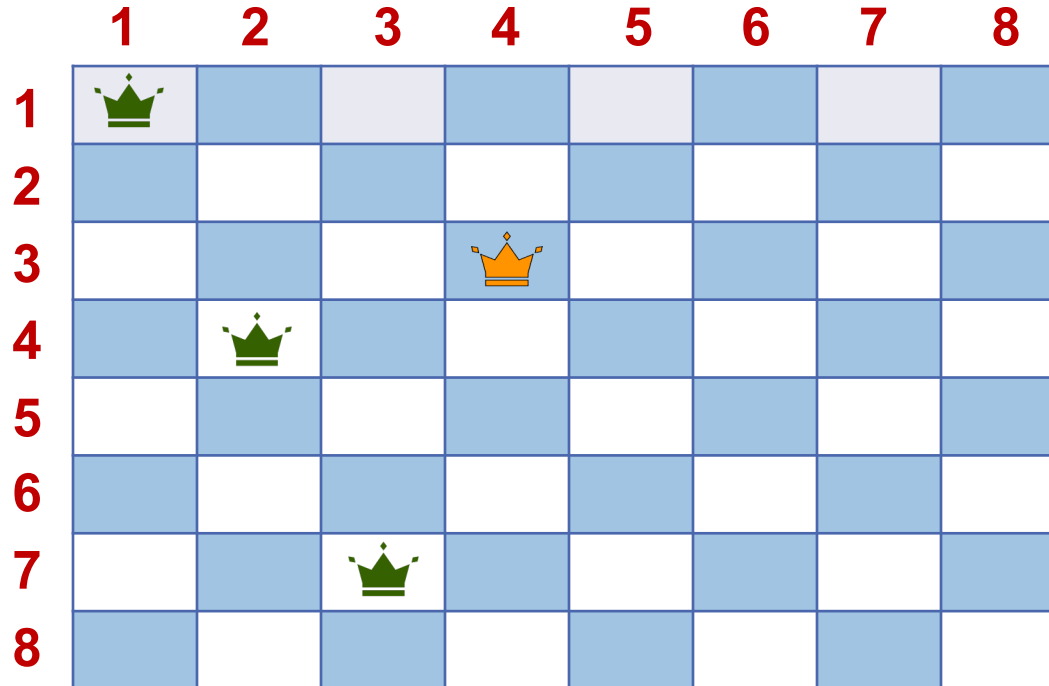


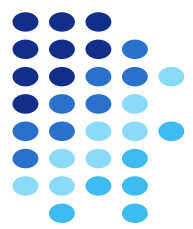
O candidato anterior é a rainha FIXA! Fazemos o **backtracking** mais uma vez!



Exemplo – 8 Rainhas

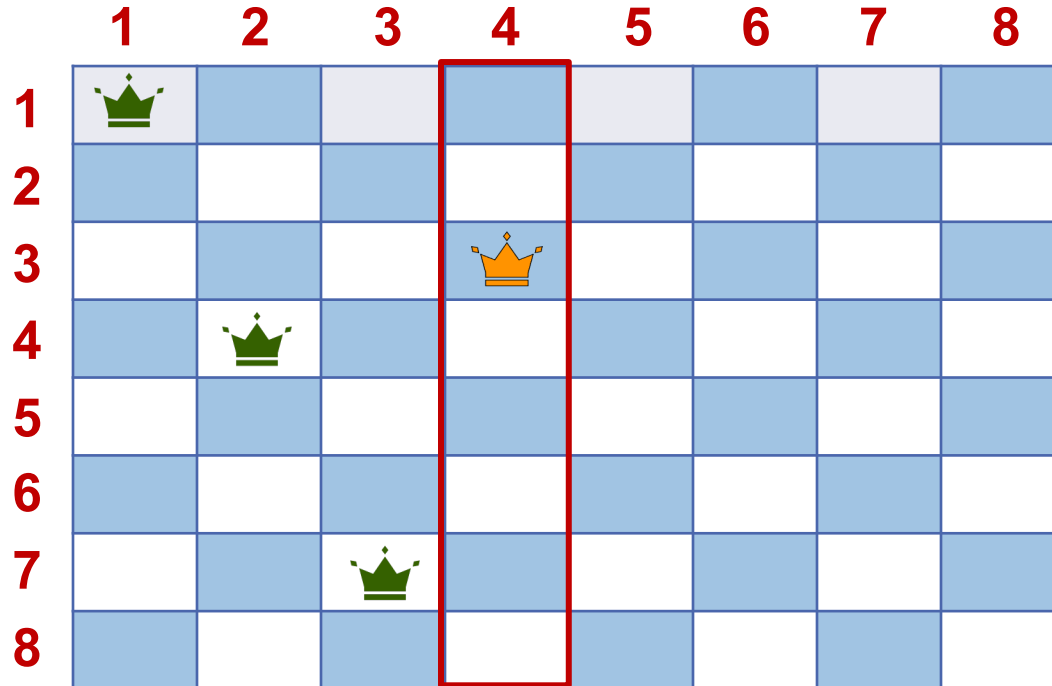
- O problema já define o local de **1 rainha FIXA**. Ex.:



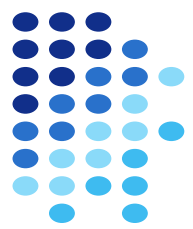


Exemplo – 8 Rainhas

- O problema já define o local de **1 rainha FIXA**. Ex.:

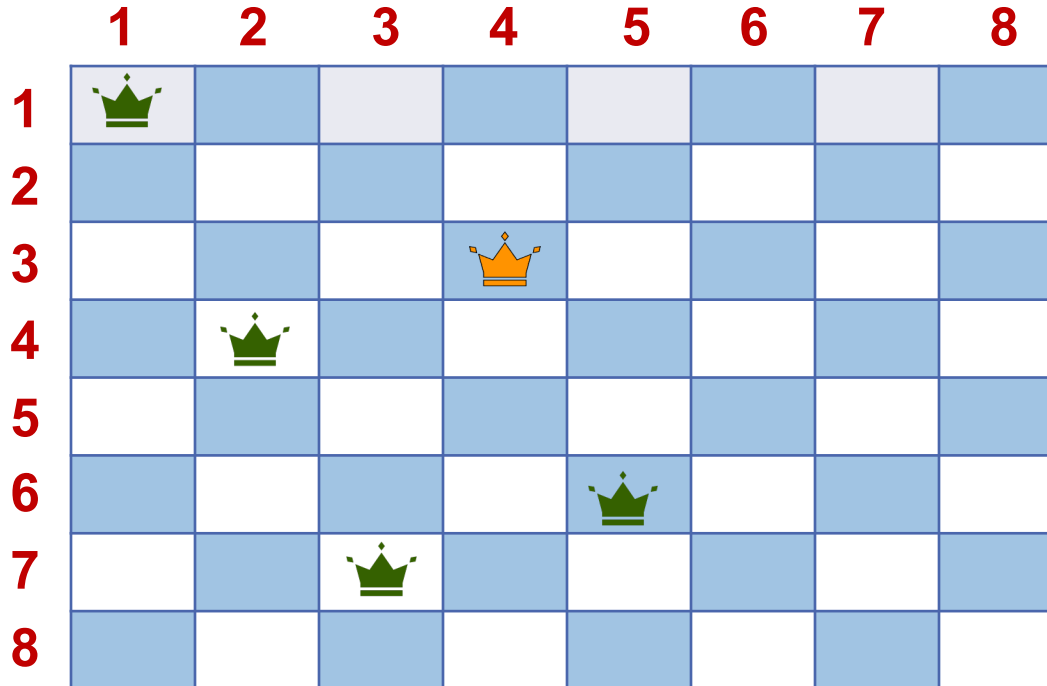


**É a rainha FIXA,
temos que pular!**

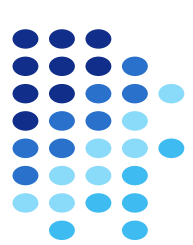


Exemplo – 8 Rainhas

- O problema já define o local de **1 rainha FIXA**. Ex.:

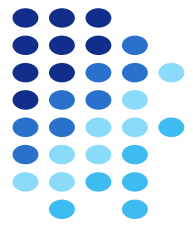


... E assim
sucessivamente!



Exemplo – 8 Rainhas

- Vamos tentar manter a lógica do algoritmo apresentado e implementar:
 - `Construct_candidates`
 - `is_a_solution`
 - `process_solution`



Exemplo – 8 Rainhas

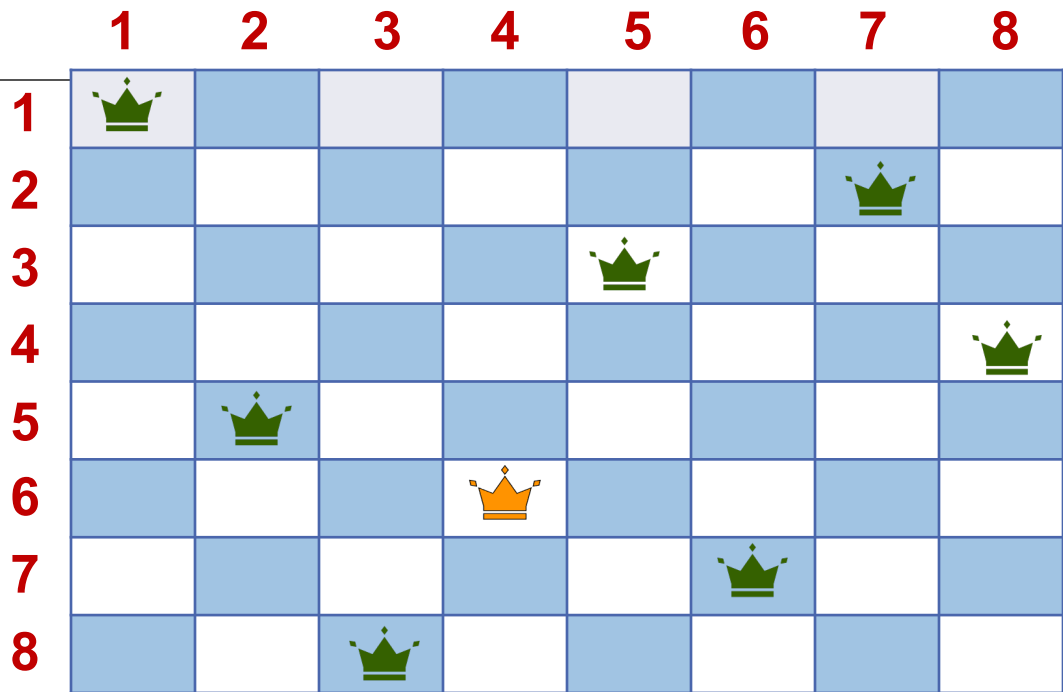
- Vamos representar a solução como um *array* (e não uma matriz)

array =

0	1	2	3	4	5	6	7	8

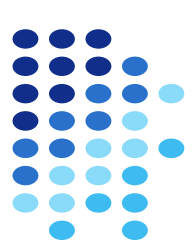
Exemplo – 8 Rainhas

- Assim, por exemplo, o *array* para representar a solução ao lado será:



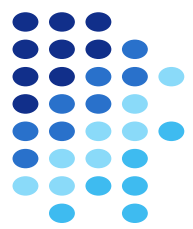
array =

0	1	2	3	4	5	6	7	8
	1	5	8	6	3	7	2	4




Exemplo – 8 Rainhas

- Como descobrir o próximo candidato (k)?
 - Não pode atacar a rainha fixa
 - Não pode estar na mesma linha das anteriores
 - Não pode estar na mesma diagonal das anteriores
 - Como saber se estão na mesma diagonal?



Exemplo – 8 Rainhas

- Como saber as diagonais?

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

Por exemplo:

Rainha Diagonais

(4,2) (1,5), (3,1), (6,4), (8,6)

O módulo da diferença entre as linhas e colunas é igual!

(4 – 1, 2 – 5) = ($|3|$, $|-3|$) = (3, 3)

(4 – 3, 2 – 1) = ($|1|$, $|1|$) = (1, 1)

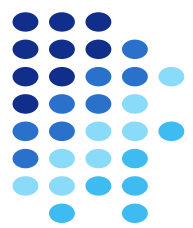
(4 – 6, 2 – 4) = ($|-2|$, $|-2|$) = (2, 2)

(4 – 8, 2 – 6) = ($|-4|$, $|-4|$) = (4, 4)


```
void constructCandidates(int a[], int k, int candidates[], int *n){
    int c, i;
    int passed = 1;
    for(c = 1; c <=8; c++){// testar os 8 candidatos
        if (!checkFixedQueen(c, k)) //verificar se o candidato ataca a rainha fixa
            continue;
        passed = 1;
        for(i = 1;i < k; i++){//verificar se o candidato ataca as rainhas anteriores
            if (a[i] == c){//mesma linha
                passed = 0;
                break;
            }
            if (abs(a[i] - c) == abs(i - k)){//mesma diagonal
                passed = 0;
                break;
            }
        }
        if (passed){
            candidates[*n] = c;
            *n += 1;
        }
    }
}
```

```
int checkFixedQueen(int c, int k){
    if (array[FIXED_COL] == c)
        return 0;
    if (abs(array[FIXED_COL] - c) == abs(FIXED_COL - k))
        return 0;
    return 1;
}

void bt(int a[], int k, int depth){
    if (depth == 8){
        process_Solution(a);
        return;
    }
    if (k == FIXED_COL){// se for a coluna da rainha fixa, pula para a próxima
        bt(a, k + 1, depth + 1);
        return;
    }
    //construir candidatos e chamar backtracking
    //...
}
```



Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. **Introduction to Algorithms, Third Edition** (3rd. ed.). The MIT Press.
- Robert Sedgewick. 2002. **Algorithms in C** (3rd. ed.). Addison-Wesley Longman Publishing Co., Inc., USA.
- Material baseado nos slides de **Rodrigo Paes**, Programação Avançada. Instituto de Computação. Universidade Federal de Alagoas (UFAL), Maceió, Brasil. (<https://sites.google.com/site/ldsicufal/disciplinas/programacao-avancada/backtracking>)
- Material baseado nos slides de **Rafael C. S. Schouery**, Estrutura de Dados. Instituto de Computação. Universidade Estadual de Campinas (UNICAMP), Campinas, Brasil. (<https://www.ic.unicamp.br/~rafael/cursos/2s2019/mc202/slides/unidade09-backtracking.pdf>)
- Material baseado nos slides de **Túlio Toffolo e Marco Antônio Carvalho**, Algoritmos e Programação Avançada. Departamento de Computação. Universidade Federal de Ouro Preto (UFOP), Ouro Preto, Brasil. (http://www3.decom.ufop.br/toffolo/site_media/uploads/2011-1/bcc402/slides/10._backtracking.pdf)
- Material baseado nos slides de **Mário César San Felice**, Algoritmos e Estruturas de Dados 1. Departamento de Computação. Universidade Federal de São Carlos (UFSCar), São Carlos, Brasil. (https://www2.dc.ufscar.br/~mario/ensino/2019s1/aed1/aula25_slides.pdf)