Recursão

MC102

Recursão

Recursão é o conceito de uma função chamar a si mesma para resolver algum problema computacional

Recursão

Recursão é o conceito de uma função chamar a si mesma para resolver algum problema computacional

- Sabemos resolver instâncias pequenas do problema

Recursão

Recursão é o conceito de uma função chamar a si mesma para resolver algum problema computacional

- Sabemos resolver instâncias pequenas do problema
  - Chamamos de **caso base**

Recursão

Recursão é o conceito de uma função chamar a si mesma para resolver algum problema computacional

- Sabemos resolver instâncias pequenas do problema
  - Chamamos de **caso base**
- Sabemos resolver uma instância maior a partir das menores

Recursão

Recursão é o conceito de uma função chamar a si mesma para resolver algum problema computacional

- Sabemos resolver instâncias pequenas do problema
  - Chamamos de **caso base**
- Sabemos resolver uma instância maior a partir das menores
  - Chamamos de **caso geral**

Recursão

Recursão é o conceito de uma função chamar a si mesma para resolver algum problema computacional

- Sabemos resolver instâncias pequenas do problema
  - Chamamos de **caso base**
- Sabemos resolver uma instância maior a partir das menores
  - Chamamos de **caso geral**

Por exemplo, para calcular $n! = \prod_{i=1}^{n} i$:

Recursão

Recursão é o conceito de uma função chamar a si mesma para resolver algum problema computacional

- Sabemos resolver instâncias pequenas do problema
  - Chamamos de **caso base**
- Sabemos resolver uma instância maior a partir das menores
  - Chamamos de **caso geral**

Por exemplo, para calcular $n! = \prod_{i=1}^{n} i$:

- **Base**: Sabemos resolver $0!$, pois $0! = 1$

Recursão

Recursão é o conceito de uma função chamar a si mesma para resolver algum problema computacional

- Sabemos resolver instâncias pequenas do problema
  - Chamamos de **caso base**
- Sabemos resolver uma instância maior a partir das menores
  - Chamamos de **caso geral**

Por exemplo, para calcular $n! = \prod_{i=1}^{n} i$:

- **Base**: Sabemos resolver $0!$, pois $0! = 1$
- **Geral**: Sabemos resolver $n!$ a partir de $(n-1)!$ pois

$$n! = \prod_{i=1}^{n} i = n \prod_{i=1}^{n-1} i = n(n-1)!$$

Exemplo: Fatorial

Calculando $n!$ recursivamente:

Exemplo: Fatorial

Calculando $n!$ recursivamente:

```
1 def fatorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * fatorial(n - 1)
```

Exemplo: Fatorial

Calculando $n!$ recursivamente:

```
1 def fatorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * fatorial(n - 1)
```

Mas como isso pode funcionar se a função chama a si mesma?

Exemplo: Fatorial

Calculando $n!$ recursivamente:

```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Mas como isso pode funcionar se a função chama a si mesma?

O Python sabe a linha de código que fez a chamada de função

Exemplo: Fatorial

Calculando $n!$ recursivamente:

```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Mas como isso pode funcionar se a função chama a si mesma?

O Python sabe a linha de código que fez a chamada de função

- Isso gera a **pilha de chamadas**

Exemplo: Fatorial

Calculando $n!$ recursivamente:

```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Mas como isso pode funcionar se a função chama a si mesma?

O Python sabe a linha de código que fez a chamada de função

- Isso gera a **pilha de chamadas**
- Quando uma função é chamada, ela vai "em cima" da atual

Exemplo: Fatorial

Calculando $n!$ recursivamente:

```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Mas como isso pode funcionar se a função chama a si mesma?

O Python sabe a linha de código que fez a chamada de função

- Isso gera a **pilha de chamadas**
- Quando uma função é chamada, ela vai "em cima" da atual

O Python sabe também qual era o valor das variáveis locais

Exemplo: Fatorial

Calculando $n!$ recursivamente:

```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

Mas como isso pode funcionar se a função chama a si mesma?

O Python sabe a linha de código que fez a chamada de função
- Isso gera a **pilha de chamadas**
- Quando uma função é chamada, ela vai "em cima" da atual

O Python sabe também qual era o valor das variáveis locais
- Ele consegue restaurar esses valores quando voltar

Exemplo: Fatorial

Calculando $n!$ recursivamente:

```
1 def fatorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * fatorial(n - 1)
```

Exemplo: Fatorial

Calculando $n!$ recursivamente:

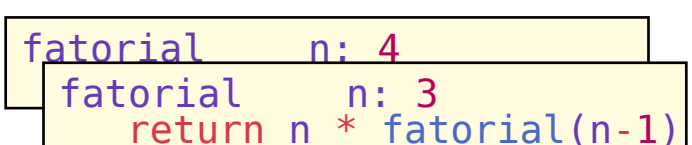
```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```

```
fatorial     n: 4
   return n * fatorial(n-1)
```

Exemplo: Fatorial

Calculando $n!$ recursivamente:

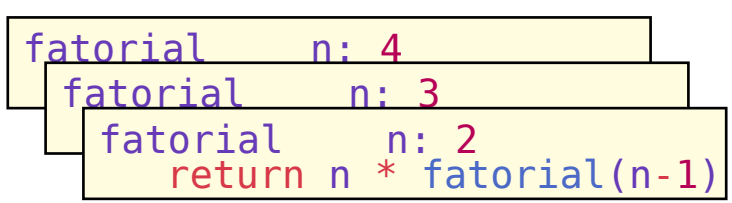
```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```



Exemplo: Fatorial

Calculando $n!$ recursivamente:

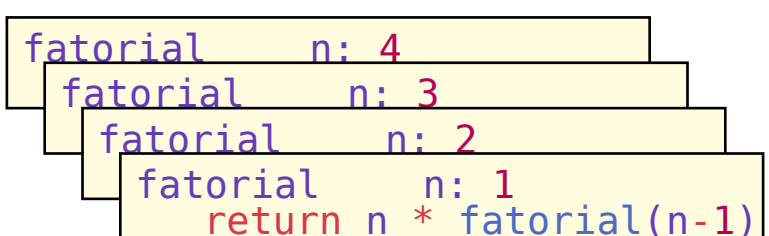
```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```



Exemplo: Fatorial

Calculando $n!$ recursivamente:

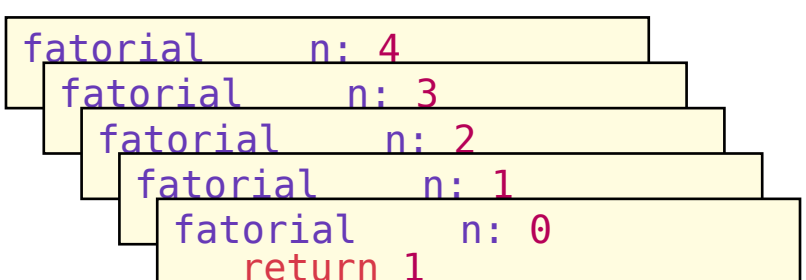
```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```



Exemplo: Fatorial

Calculando $n!$ recursivamente:

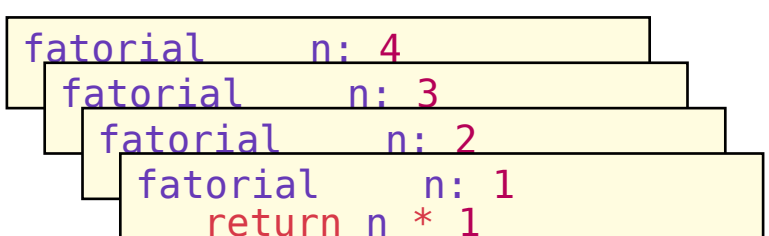
```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```



Exemplo: Fatorial

Calculando $n!$ recursivamente:

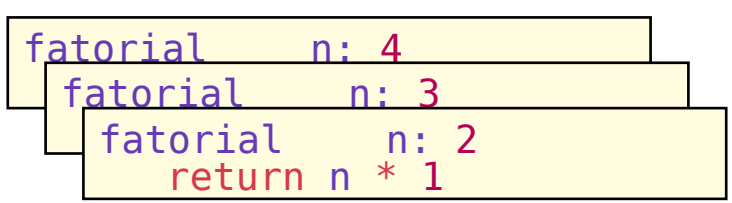
```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```



Exemplo: Fatorial

Calculando $n!$ recursivamente:

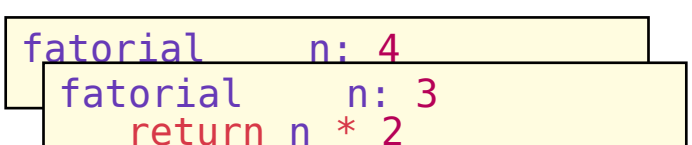
```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```



Exemplo: Fatorial

Calculando $n!$ recursivamente:

```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)
```



Exemplo: Fatorial

Calculando $n!$ recursivamente:

```
1 def fatorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * fatorial(n - 1)
```

```
fatorial     n: 4
   return n * 6
```

Exemplo: Fatorial

Calculando $n!$ recursivamente:

```
1 def fatorial(n):
2     if n == 0:
3         return 1
4     else:
5         return n * fatorial(n - 1)
```

```
fatorial     n: 4
   return n * 6
```

Exemplo: Último termo da PA

Uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_i = a_{i-1} + r$, para todo $1 < i \leq n$

Exemplo: Último termo da PA

Uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_i = a_{i-1} + r$, para todo $1 < i \leq n$

Queremos um algoritmo recursivo para calcular a_n

Exemplo: Último termo da PA

Uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_i = a_{i-1} + r$, para todo $1 < i \leq n$

Queremos um algoritmo recursivo para calcular a_n

- **Base**:

Exemplo: Último termo da PA

Uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_i = a_{i-1} + r$, para todo $1 < i \leq n$

Queremos um algoritmo recursivo para calcular a_n

- **Base**: Se $n = 1$, $a_n = a_1$

Exemplo: Último termo da PA

Uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_i = a_{i-1} + r$, para todo $1 < i \leq n$

Queremos um algoritmo recursivo para calcular a_n

- **Base**: Se $n = 1$, $a_n = a_1$
- **Geral**:

Exemplo: Último termo da PA

Uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_i = a_{i-1} + r$, para todo $1 < i \leq n$

Queremos um algoritmo recursivo para calcular a_n

- **Base**: Se $n = 1$, $a_n = a_1$
- **Geral**: Se $n > 1$, $a_n = a_{n-1} + r$

Exemplo: Último termo da PA

Uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_i = a_{i-1} + r$, para todo $1 < i \leq n$

Queremos um algoritmo recursivo para calcular a_n

- **Base**: Se $n = 1$, $a_n = a_1$
- **Geral**: Se $n > 1$, $a_n = a_{n-1} + r$
  - Estamos indo de uma instância maior para uma menor

Exemplo: Último termo da PA

Uma Progressão Aritmética (PA)

- é uma sequência de números $(a_1, a_2, ..., a_n)$
- onde existe um número r tal que
- $a_i = a_{i-1} + r$, para todo $1 < i \leq n$

Queremos um algoritmo recursivo para calcular a_n

- **Base**: Se $n = 1$, $a_n = a_1$
- **Geral**: Se $n > 1$, $a_n = a_{n-1} + r$
  - Estamos indo de uma instância maior para uma menor

Solução:

```python
def termo(a_1, r, n):
    '''Calcula o n-ésimo termo de uma PA
    iniciando em a_1 com razão r'''
    if n == 1:
        return a_1
    else:
        return r + termo(a_1, r, n - 1)
```

Exemplo: Progressão aritmética com impressão

E se quisermos imprimir a progressão aritmética?

Exemplo: Progressão aritmética com impressão

E se quisermos imprimir a progressão aritmética?

```
1 def termo(a_1, r, n):
2     if n == 1:
3         print(a_1)
4         return a_1
5     else:
6         atual = r + termo(a_1, r, n - 1)
7         print(atual)
8         return atual
```

Exemplo: Progressão aritmética com impressão

E se quisermos imprimir a progressão aritmética?

```
1 def termo(a_1, r, n):
2     if n == 1:
3         print(a_1)
4         return a_1
5     else:
6         atual = r + termo(a_1, r, n - 1)
7         print(atual)
8         return atual
```

Vamos depurar!

Exemplo: Fibonnaci

A sequência de Fibonnaci é: $1, 1, 2, 3, 5, 8, \ldots$

Exemplo: Fibonnaci

A sequência de Fibonnaci é: 1, 1, 2, 3, 5, 8, ...

- Começa com 1, 1

Exemplo: Fibonnaci

A sequência de Fibonnaci é: $1, 1, 2, 3, 5, 8, \ldots$

- Começa com 1, 1
- Cada elemento a seguir é a soma dos dois anteriores

Exemplo: Fibonnaci

A sequência de Fibonnaci é: $1, 1, 2, 3, 5, 8, \ldots$

- Começa com 1, 1
- Cada elemento a seguir é a soma dos dois anteriores

Ou seja,

$$f(n) = \begin{cases} f(n-1) + f(n-2) \text{ se } n > 2 \\ 1 \text{ se } n = 1 \text{ ou } n = 2 \end{cases}$$

Exemplo: Fibonnaci

A sequência de Fibonnaci é: $1, 1, 2, 3, 5, 8, \ldots$

- Começa com 1, 1
- Cada elemento a seguir é a soma dos dois anteriores

Ou seja,

$$f(n) = \begin{cases} f(n-1) + f(n-2) \text{ se } n > 2 \\ 1 \text{ se } n = 1 \text{ ou } n = 2 \end{cases}$$

É o que chamamos na matemática de **recorrência**

- Uma função definida recursivamente
- $n!$ é outro exemplo de recorrência

Exemplo: Fibonnaci

A sequência de Fibonnaci é: $1, 1, 2, 3, 5, 8, \ldots$

- Começa com 1, 1
- Cada elemento a seguir é a soma dos dois anteriores

Ou seja,

$$f(n) = \begin{cases} f(n-1) + f(n-2) \text{ se } n > 2 \\ 1 \text{ se } n = 1 \text{ ou } n = 2 \end{cases}$$

É o que chamamos na matemática de **recorrência**

- Uma função definida recursivamente
- $n!$ é outro exemplo de recorrência

Note que temos dois casos bases e um caso geral

Exemplo: Fibonnaci

A sequência de Fibonnaci é: $1, 1, 2, 3, 5, 8, \ldots$

- Começa com 1, 1
- Cada elemento a seguir é a soma dos dois anteriores

Ou seja,

$$f(n) = \begin{cases} f(n-1) + f(n-2) \text{ se } n > 2 \\ 1 \text{ se } n = 1 \text{ ou } n = 2 \end{cases}$$

É o que chamamos na matemática de **recorrência**

- Uma função definida recursivamente
- $n!$ é outro exemplo de recorrência

Note que temos dois casos bases e um caso geral

- E que resolvemos uma instância a partir de duas menores

Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```

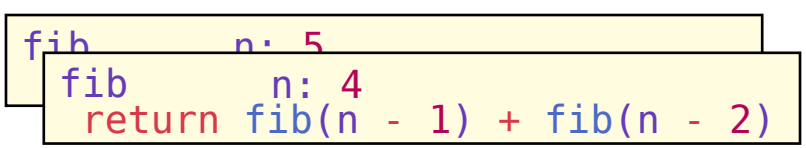
Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```

```
fib       n: 5
 return fib(n - 1) + fib(n - 2)
```

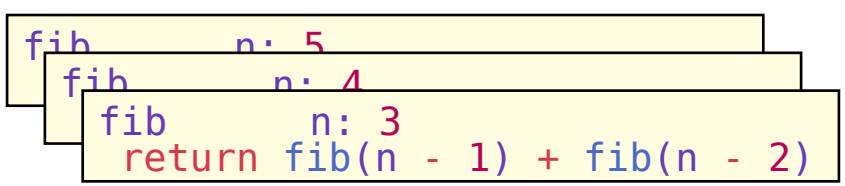
Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```



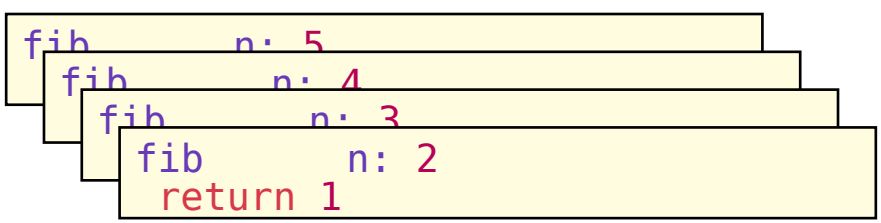
Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```



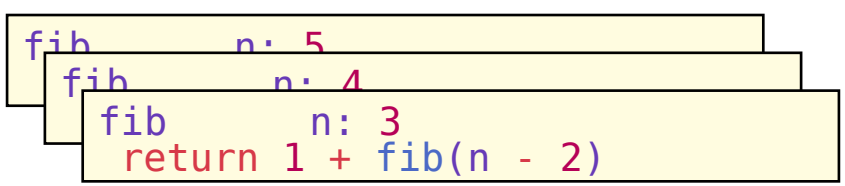
Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```



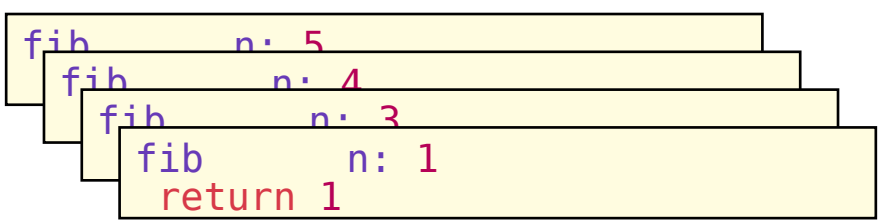
Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```



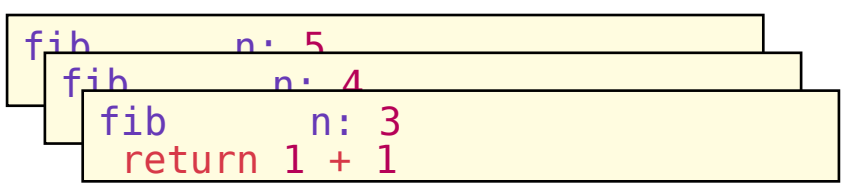
Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```



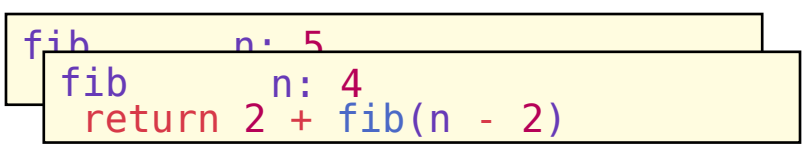
Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```



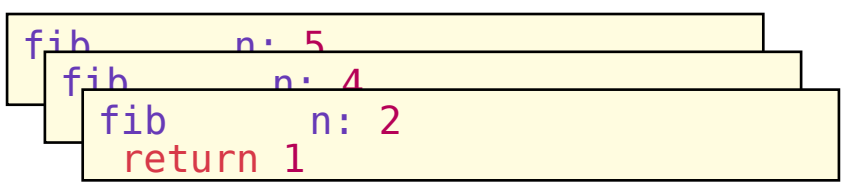
Exemplo: Fibonnaci

```python
def fib(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```



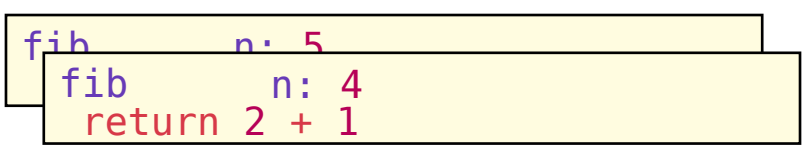
Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```



Exemplo: Fibonnaci

```python
def fib(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```



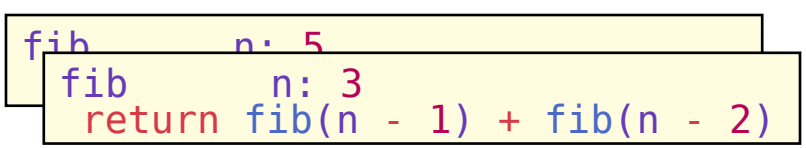
Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```

```
fib       n: 5
 return 3 + fib(n - 2)
```

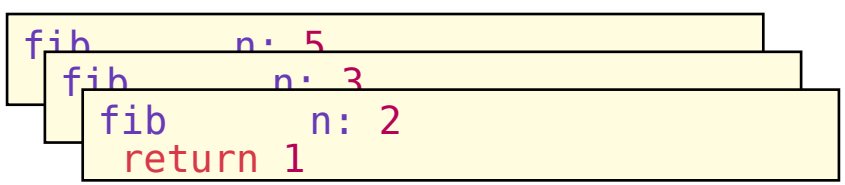
Exemplo: Fibonnaci

```python
def fib(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```



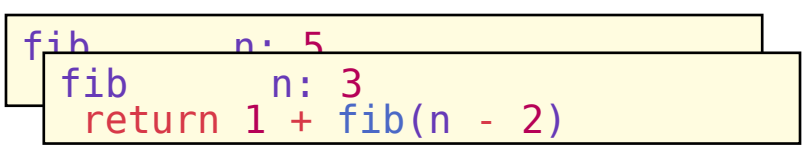
Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```



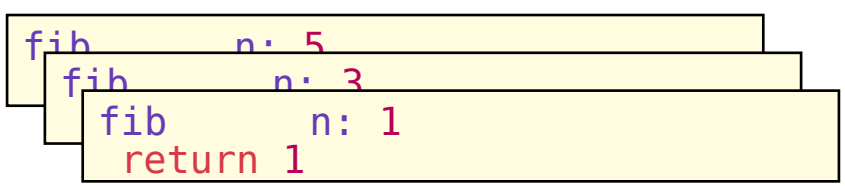
Exemplo: Fibonnaci

```python
def fib(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```



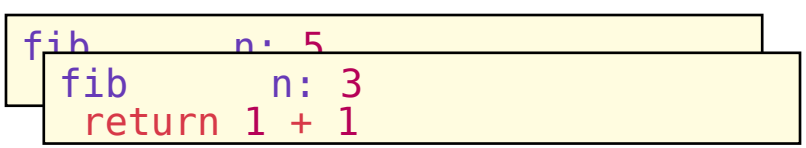
Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```



Exemplo: Fibonnaci

```python
def fib(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```



Exemplo: Fibonnaci

```
1 def fib(n):
2     if n == 1 or n == 2:
3         return 1
4     else:
5         return fib(n - 1) + fib(n - 2)
```

```
fib       n: 5
 return 3 + 2
```

Backtracking

O back e o tracking

Força Bruta: tentar todas as soluções possíveis, selecionando as viáveis.

O back e o tracking

Força Bruta: tentar todas as soluções possíveis, selecionando as viáveis.

Exemplo: Três estudantes: Alice, Bruno, Carlos que precisam se sentar em 3 cadeiras.

O back e o tracking

Força Bruta: tentar todas as soluções possíveis, selecionando as viáveis.

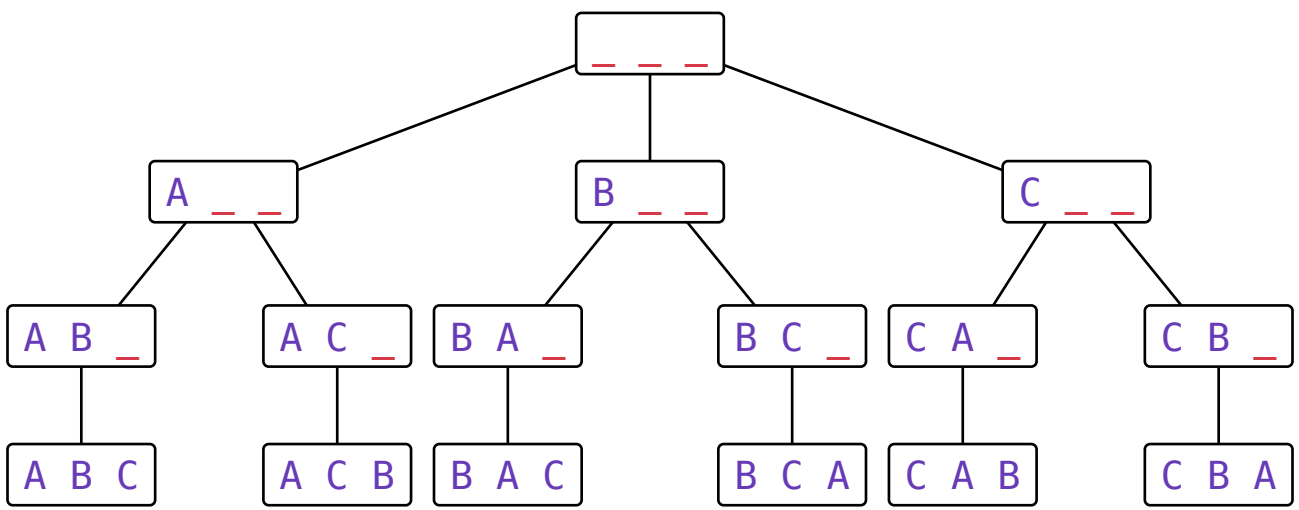
Exemplo: Três estudantes: Alice, Bruno, Carlos que precisam se sentar em 3 cadeiras. Quantas soluções sõa possíveis?

O back e o tracking

Força Bruta: tentar todas as soluções possíveis, selecionando as viáveis.

Exemplo: Três estudantes: Alice, Bruno, Carlos que precisam se sentar em 3 cadeiras. Quantas soluções sõa possíveis? n! = 3! = 6

Árvore de Decisão

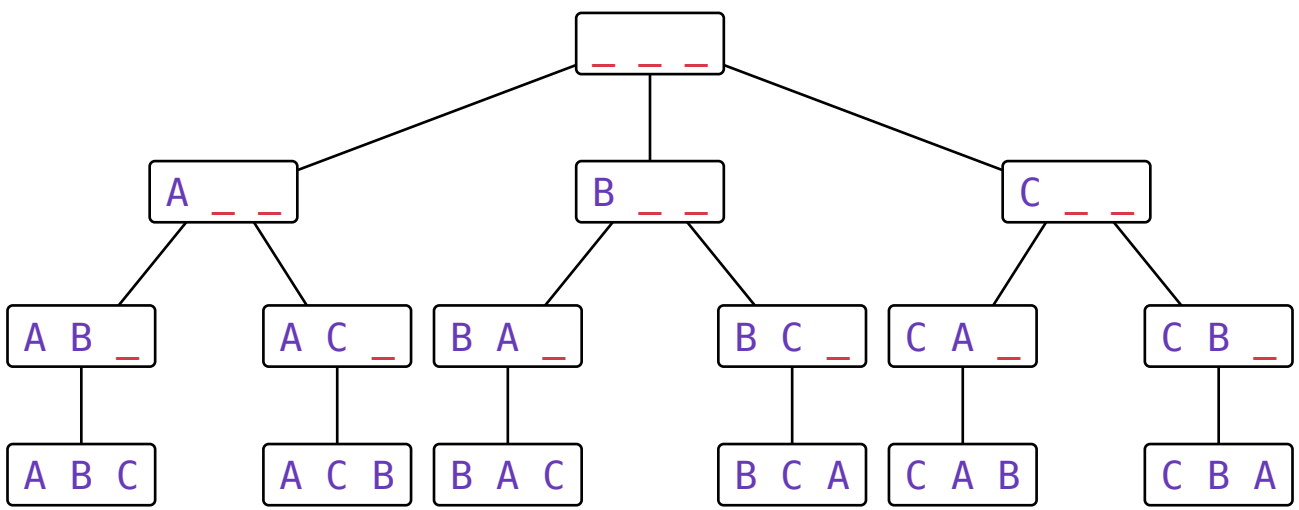


O back e o tracking

Força Bruta: tentar todas as soluções possíveis, selecionando as viáveis.

Exemplo: Três estudantes: Alice, Bruno, Carlos que precisam se sentar em 3 cadeiras. Quantas soluções sõa possíveis? n! = 3! = 6

Árvore de Decisão

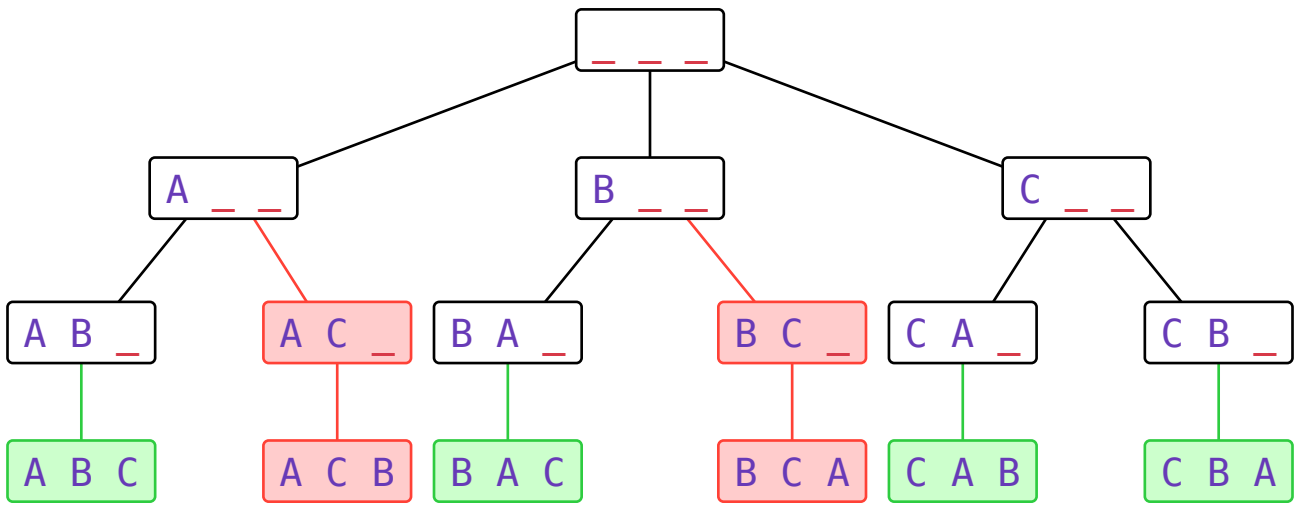


O back e o tracking

Restrições: Carlos não pode se sentar entre Alice e Bruno.

O back e o tracking

Restrições: Carlos não pode se sentar entre Alice e Bruno.



Ideia central do backtracking

- Gerar soluções parciais

Ideia central do backtracking

- Gerar soluções parciais
- Verificar se a solução parcial é viável
- Se for, continuar gerando a partir dela

Ideia central do backtracking

- Gerar soluções parciais
- Verificar se a solução parcial é viável
- Se for, continuar gerando a partir dela
- Se não for, descartar e voltar para a etapa anterior (backtrack)

Ideia central do backtracking

- Gerar soluções parciais
- Verificar se a solução parcial é viável
- Se for, continuar gerando a partir dela
- Se não for, descartar e voltar para a etapa anterior (backtrack)
- Repetir até encontrar a solução completa ou esgotar as possibilidades

Ideia central do backtracking

- Gerar soluções parciais
- Verificar se a solução parcial é viável
- Se for, continuar gerando a partir dela
- Se não for, descartar e voltar para a etapa anterior (backtrack)
- Repetir até encontrar a solução completa ou esgotar as possibilidades

Subconjuntos

</> Code

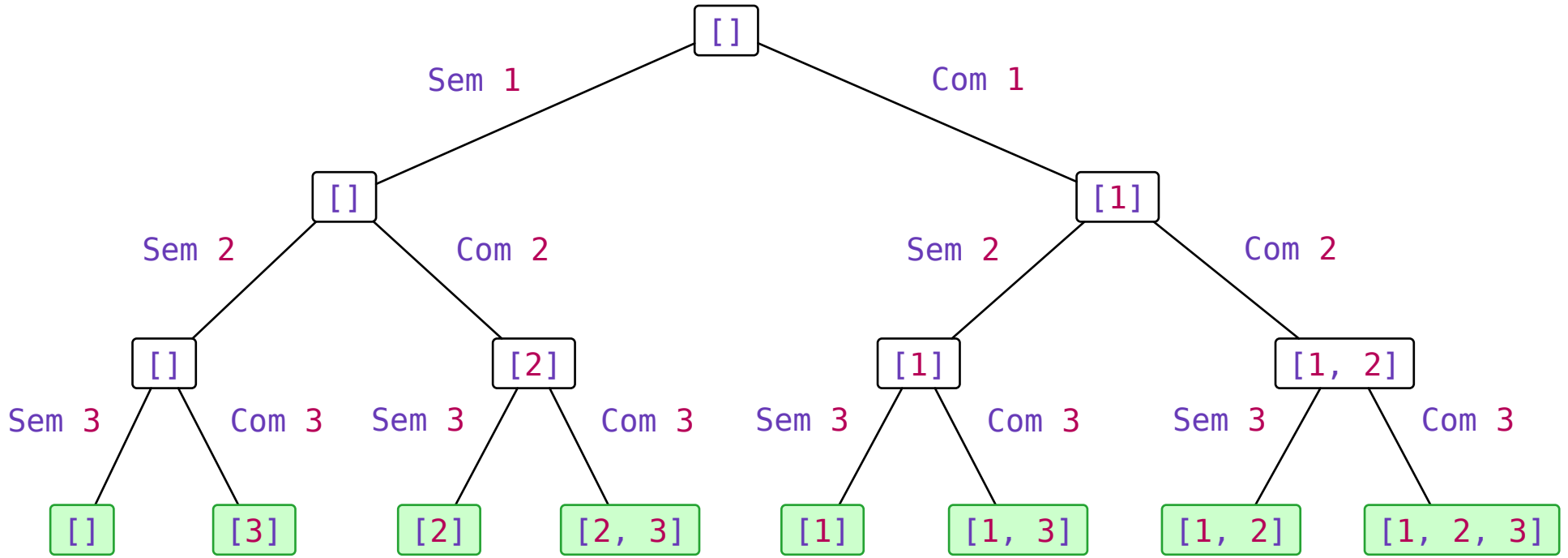
Dado uma lista de elementos, retorne todos os subconjuntos desta lista.

Subconjuntos

</> Code

Dado uma lista de elementos, retorne todos os subconjuntos desta lista.

Exemplo para a lista `[1, 2, 3]`:


[]
Sem 1
Com 1
[]
[1]
Sem 2
Com 2
Sem 2
Com 2
[]
[2]
[1]
[1, 2]
Sem 3
Com 3
Sem 3
Com 3
Sem 3
Com 3
Sem 3
Com 3
[]
[3]
[2]
[2, 3]
[1]
[1, 3]
[1, 2]
[1, 2, 3]

Subconjuntos

```python
solutions = []
def subset(lista, index=0, path=[]):
    if index == len(lista):
        solutions.append(path)
        return

    # Não incluir o elemento atual
    subset(lista, index + 1, path)

    # Incluir o elemento atual
    subset(lista, index + 1, path + [lista[index]])
```

Ideia central do backtracking

Todos os problemas de backtracking podem ser resolvidos em 4 etapas:

- Caso base

Ideia central do backtracking

Todos os problemas de backtracking podem ser resolvidos em 4 etapas:

- Caso base
- Escolhas (recursão)

Ideia central do backtracking

Todos os problemas de backtracking podem ser resolvidos em 4 etapas:

- Caso base
- Escolhas (recursão)
- Restrições (validação)

Ideia central do backtracking

Todos os problemas de backtracking podem ser resolvidos em 4 etapas:

- Caso base
- Escolhas (recursão)
- Restrições (validação)
- Backtrack

Ideia central do backtracking

Todos os problemas de backtracking podem ser resolvidos em 4 etapas:

- Caso base
- Escolhas (recursão)
- Restrições (validação)
- Backtrack

Exemplo do labirinto

- Imagine um labirinto representado por uma matriz, onde 0s são caminhos livres

Exemplo do labirinto

- Imagine um labirinto representado por uma matriz, onde 0s são caminhos livres
- Queremos encontrar um caminho do ponto A ao ponto B

Exemplo do labirinto

- Imagine um labirinto representado por uma matriz, onde 0s são caminhos livres
- Queremos encontrar um caminho do ponto A ao ponto B
- Podemos usar backtracking para explorar o labirinto:

Exemplo do labirinto

- Imagine um labirinto representado por uma matriz, onde 0s são caminhos livres
- Queremos encontrar um caminho do ponto A ao ponto B
- Podemos usar backtracking para explorar o labirinto:
  - Começamos no ponto A

Exemplo do labirinto

- Imagine um labirinto representado por uma matriz, onde 0s são caminhos livres
- Queremos encontrar um caminho do ponto A ao ponto B
- Podemos usar backtracking para explorar o labirinto:
  - Começamos no ponto A
  - Em cada passo, tentamos mover para uma direção (cima, baixo, esquerda, direita)

Exemplo do labirinto

- Imagine um labirinto representado por uma matriz, onde 0s são caminhos livres
- Queremos encontrar um caminho do ponto A ao ponto B
- Podemos usar backtracking para explorar o labirinto:
  - Começamos no ponto A
  - Em cada passo, tentamos mover para uma direção (cima, baixo, esquerda, direita)
  - Se o movimento for válido (dentro dos limites e sem paredes), continuamos a partir dessa nova posição

Exemplo do labirinto

- Imagine um labirinto representado por uma matriz, onde 0s são caminhos livres
- Queremos encontrar um caminho do ponto A ao ponto B
- Podemos usar backtracking para explorar o labirinto:
  - Começamos no ponto A
  - Em cada passo, tentamos mover para uma direção (cima, baixo, esquerda, direita)
  - Se o movimento for válido (dentro dos limites e sem paredes), continuamos a partir dessa nova posição
  - Se chegarmos ao ponto B, encontramos uma solução

Exemplo do labirinto

- Imagine um labirinto representado por uma matriz, onde 0s são caminhos livres
- Queremos encontrar um caminho do ponto A ao ponto B
- Podemos usar backtracking para explorar o labirinto:
  - Começamos no ponto A
  - Em cada passo, tentamos mover para uma direção (cima, baixo, esquerda, direita)
  - Se o movimento for válido (dentro dos limites e sem paredes), continuamos a partir dessa nova posição
  - Se chegarmos ao ponto B, encontramos uma solução
  - Se não houver mais movimentos válidos, voltamos para a posição anterior e tentamos outra direção (backtrack)

Exemplo do labirinto

- Imagine um labirinto representado por uma matriz, onde 0s são caminhos livres
- Queremos encontrar um caminho do ponto A ao ponto B
- Podemos usar backtracking para explorar o labirinto:
  - Começamos no ponto A
  - Em cada passo, tentamos mover para uma direção (cima, baixo, esquerda, direita)
  - Se o movimento for válido (dentro dos limites e sem paredes), continuamos a partir dessa nova posição
  - Se chegarmos ao ponto B, encontramos uma solução
  - Se não houver mais movimentos válidos, voltamos para a posição anterior e tentamos outra direção (backtrack)
  - Repetimos esse processo até encontrar o caminho ou esgotar as possibilidades

Exemplo do labirinto

- Imagine um labirinto representado por uma matriz, onde 0s são caminhos livres
- Queremos encontrar um caminho do ponto A ao ponto B
- Podemos usar backtracking para explorar o labirinto:
  - Começamos no ponto A
  - Em cada passo, tentamos mover para uma direção (cima, baixo, esquerda, direita)
  - Se o movimento for válido (dentro dos limites e sem paredes), continuamos a partir dessa nova posição
  - Se chegarmos ao ponto B, encontramos uma solução
  - Se não houver mais movimentos válidos, voltamos para a posição anterior e tentamos outra direção (backtrack)
  - Repetimos esse processo até encontrar o caminho ou esgotar as possibilidades

Exemplo n rainhas

</> Code

Dado um tabuleiro de xadrez de tamanho n x n, queremos colocar n rainhas no tabuleiro de forma que nenhuma rainha ataque a outra.

Exemplo n rainhas

</> Code

Dado um tabuleiro de xadrez de tamanho n x n, queremos colocar n rainhas no tabuleiro de forma que nenhuma rainha ataque a outra.

- Escolhas: Uma rainha por coluna; precisamos escolher uma linha para cada

Exemplo n rainhas

</> Code

Dado um tabuleiro de xadrez de tamanho n x n, queremos colocar n rainhas no tabuleiro de forma que nenhuma rainha ataque a outra.

- Escolhas: Uma rainha por coluna; precisamos escolher uma linha para cada
- Restrições:

Exemplo n rainhas

</> Code

Dado um tabuleiro de xadrez de tamanho n x n, queremos colocar n rainhas no tabuleiro de forma que nenhuma rainha ataque a outra.

- Escolhas: Uma rainha por coluna; precisamos escolher uma linha para cada
- Restrições: Nenhuma rainha pode estar na mesma linha ou diagonal que outra

Exemplo n rainhas

</> Code

Dado um tabuleiro de xadrez de tamanho n x n, queremos colocar n rainhas no tabuleiro de forma que nenhuma rainha ataque a outra.

- Escolhas: Uma rainha por coluna; precisamos escolher uma linha para cada
- Restrições: Nenhuma rainha pode estar na mesma linha ou diagonal que outra
- Caso base:

Exemplo n rainhas

</> Code

Dado um tabuleiro de xadrez de tamanho n x n, queremos colocar n rainhas no tabuleiro de forma que nenhuma rainha ataque a outra.

- Escolhas: Uma rainha por coluna; precisamos escolher uma linha para cada
- Restrições: Nenhuma rainha pode estar na mesma linha ou diagonal que outra
- Caso base: Se colocamos todas as n rainhas, encontramos uma solução

Exemplo n rainhas

</> Code

Dado um tabuleiro de xadrez de tamanho n x n, queremos colocar n rainhas no tabuleiro de forma que nenhuma rainha ataque a outra.

- Escolhas: Uma rainha por coluna; precisamos escolher uma linha para cada
- Restrições: Nenhuma rainha pode estar na mesma linha ou diagonal que outra
- Caso base: Se colocamos todas as n rainhas, encontramos uma solução
- Backtrack:

Exemplo n rainhas

</> Code

Dado um tabuleiro de xadrez de tamanho n x n, queremos colocar n rainhas no tabuleiro de forma que nenhuma rainha ataque a outra.

- Escolhas: Uma rainha por coluna; precisamos escolher uma linha para cada
- Restrições: Nenhuma rainha pode estar na mesma linha ou diagonal que outra
- Caso base: Se colocamos todas as n rainhas, encontramos uma solução
- Backtrack: Se em algum momento não conseguimos colocar uma rainha sem violar as restrições, voltamos para a coluna anterior e tentamos outra linha para a rainha daquela coluna.

Exemplo n rainhas

</> Code

Dado um tabuleiro de xadrez de tamanho n x n, queremos colocar n rainhas no tabuleiro de forma que nenhuma rainha ataque a outra.

- Escolhas: Uma rainha por coluna; precisamos escolher uma linha para cada
- Restrições: Nenhuma rainha pode estar na mesma linha ou diagonal que outra
- Caso base: Se colocamos todas as n rainhas, encontramos uma solução
- Backtrack: Se em algum momento não conseguimos colocar uma rainha sem violar as restrições, voltamos para a coluna anterior e tentamos outra linha para a rainha daquela coluna.

Exemplo n rainhas

Exemplo n rainhas

```python
tabuleiro = [ ['.' for _ in range(n)] for _ in range(n) ]

def rainha(n, tabuleiro, linha=0):
    if linha == n:
        print(tabuleiro)  # Encontramos uma solução
        return

    for coluna in range(n):
        if is_valid(tabuleiro, linha, coluna):
            tabuleiro[linha][coluna] = 'Q'   # Coloca a rainha
            rainha(n, tabuleiro, linha + 1)  # Vai para a próxima linha
            tabuleiro[linha][coluna] = '.'   # Remove a rainha (backtrack)
```

Exemplo n rainhas

```python
def is_valid(tabuleiro, linha, coluna):
    # Verificar a coluna
    for i in range(len(tabuleiro)):
        if tabuleiro[i][coluna] == 'Q':
            return False
    # Verificar a diagonal superior esquerda
    for i, j in zip(range(linha, -1, -1), range(coluna, -1, -1)):
        if tabuleiro[i][j] == 'Q':
            return False
    # Verificar a diagonal superior direita
    for i, j in zip(range(linha, -1, -1), range(coluna, len(tabuleiro))):
        if tabuleiro[i][j] == 'Q':
            return False
    return True
```

Exemplo n rainhas

- Como paramos na primeira vez que encontrarmos uma solução?
  - Digamos que só estamos interessados em saber se existe.

```python
def rainha(n, tabuleiro, linha=0) -> bool:
    if linha == n:
        print(tabuleiro)  # Encontramos uma solução
        return True
    for coluna in range(n):
        if is_valid(tabuleiro, linha, coluna):
            tabuleiro[linha][coluna] = 'Q'      # Coloca a rainha
            if rainha(n, tabuleiro, linha + 1): # Se há solução, show!
                return True
            tabuleiro[linha][coluna] = '.'      # Remove a rainha (backtrack)
    return False
```

Cuidado com listas!

- Diferente de variáveis comúns, listas são mutáveis
- Se passamos uma lista para a função recursiva e modificamos ela, isso pode afetar outras chamadas recursivas que usam a mesma lista
- Para evitar isso, podemos
  - criar uma cópia da lista para cada chamada recursiva,
  - ou **restaurar** a lista para o estado anterior após a chamada recursiva (backtrack)