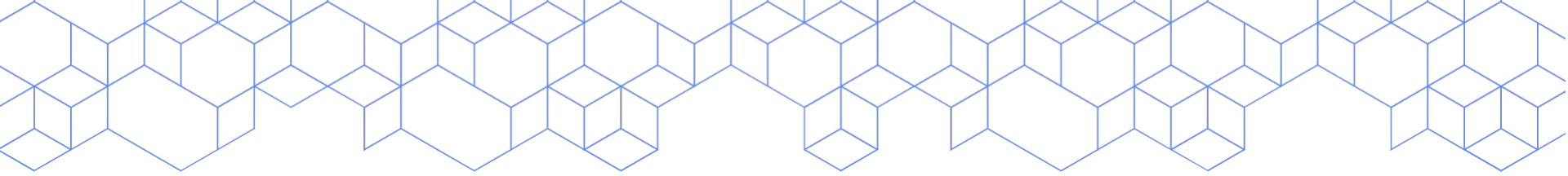


```
[algoritmo.info for algoritmo in algoritmos.sort()]
```



# **Problemas clássicos & soluções interessantes em Python**

# Olá

- Rebeca Sarai 
- Recife, Brasil     
- Formada em Engenharia da Computação 
- Aluna de mestrado da [Universidade de Pernambuco](https://www.ufpe.br/)
- Torcedora do Náutico  

✉ [rebeca@vinta.com.br](mailto:rebeca@vinta.com.br)

🐦 @\_rebecasarai

🐙 /rsarai

A Vinta é um **pequeno time de experts**

Fazemos **estratégia, design** e **desenvolvimento** para produtos web.

Tocamos projetos de **médio prazo** (1 a 3 anos) para startups que já começaram a decolar.

[vintasoftware.com](https://vintasoftware.com)

[\*\*@vintasoftware\*\*](#)

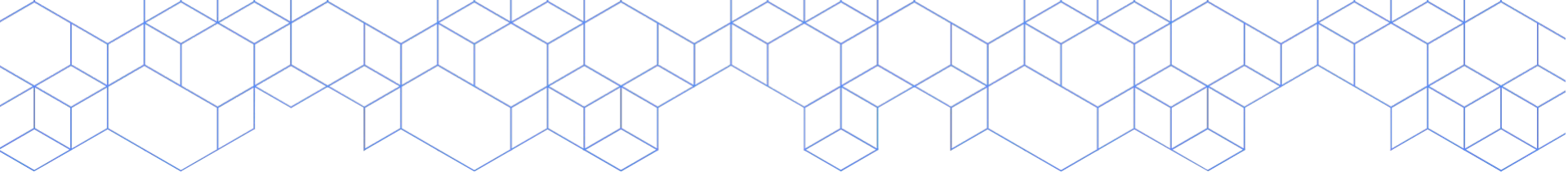




# Quer levar um pouquinho da Vinta pra casa?

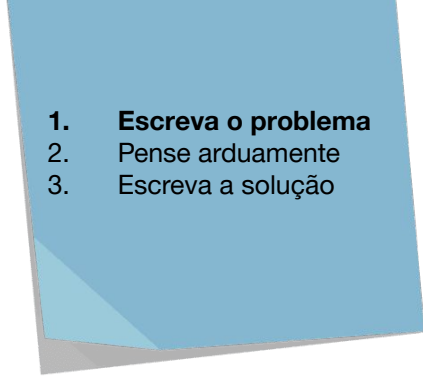
Faça um post no twitter ou instagram usando a **#VintaPyBR** e marcando a **@vintasoftware** pra ganhar esse kit arretado!

*\*enquanto durarem os estoques*



# O algoritmo de Feynman:

1. Escreva o problema.
2. Pense arduamente.
3. Escreva a solução.

- 
1. **Escreva o problema**
  2. Pense arduamente
  3. Escreva a solução

**Problema :**  
Como ordenar listas?

1. Escreva o problema
2. **Pense arduamente**
3. Escreva a solução

# Busca Sequencial

Sequential search

steps: 0

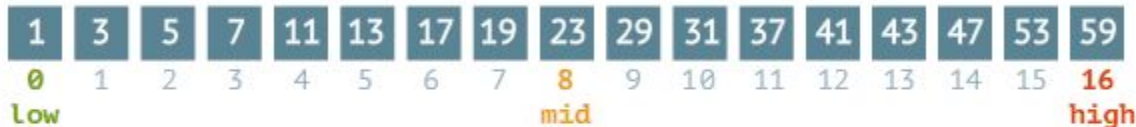


1	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16



# Busca Binária

steps: 0

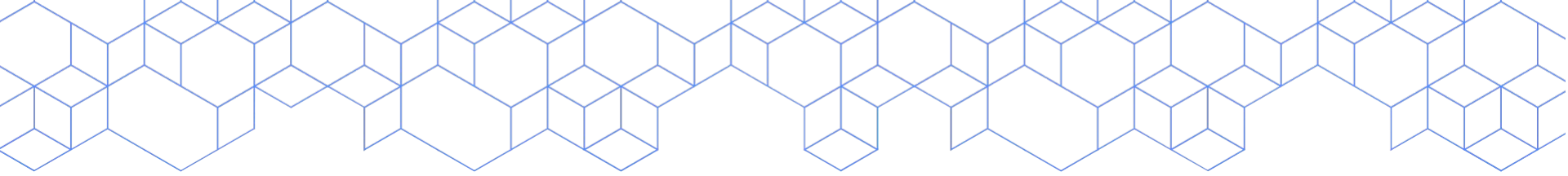


Requer uma lista ordenada. Divide a lista ao meio até que o item procurado seja encontrado

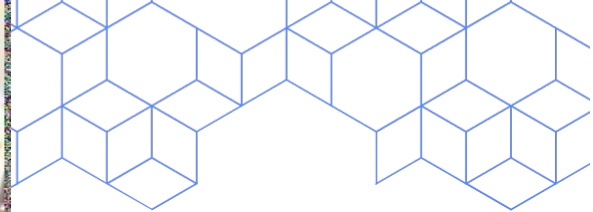
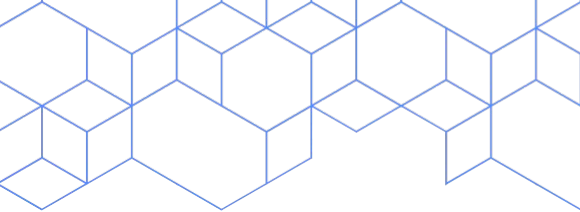
SIMPLE SEARCH	BINARY SEARCH
100 ITEMS ↓ 100 GUESSES	100 ITEMS ↓ 7 GUESSES
4,000,000,000 ITEMS ↓ 4,000,000,000 GUESSES	4,000,000,000 ITEMS ↓ 32 GUESSES
$O(n)$	$O(\log n)$
LINEAR TIME	LOGARITHMIC TIME

*O BIG SAVINGS!*

*O BIG SAVINGS!*

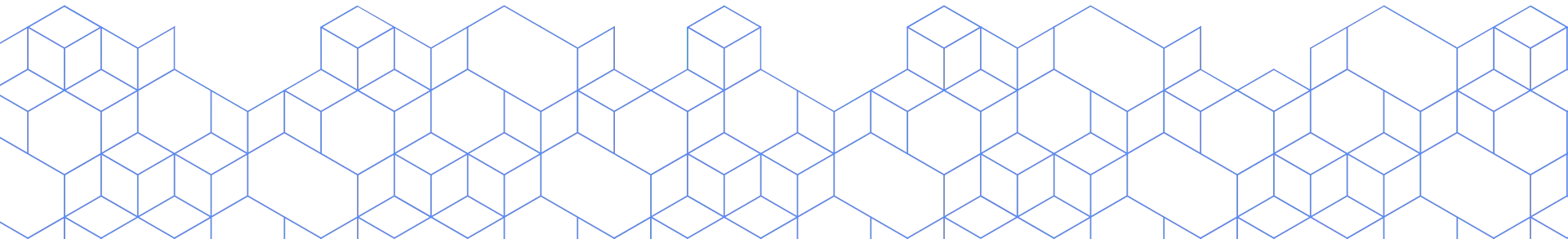


- Constant:  $O(1)$
- Logarítimo:  $O(\log n)$
- Sublinear:  $O(n^d)$  for  $d < 1$
- Linear:  $O(n)$
- Linearitimo:  $O(n \log n)$
- Quadrático:  $O(n^2)$
- Exponencial:  $O(2^n)$

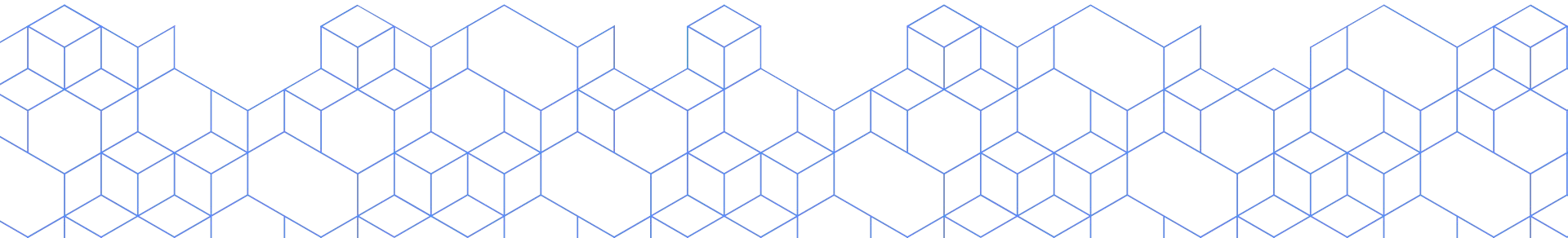
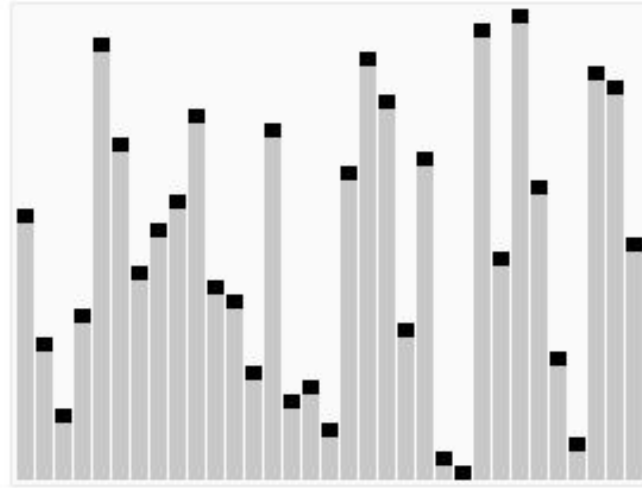


# Ordenação por Inserção

6 5 3 1 8 7 2 4

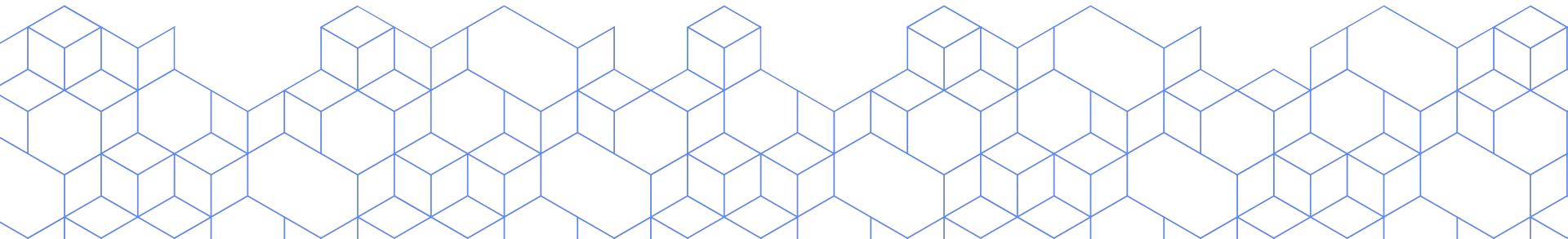


# Quicksort



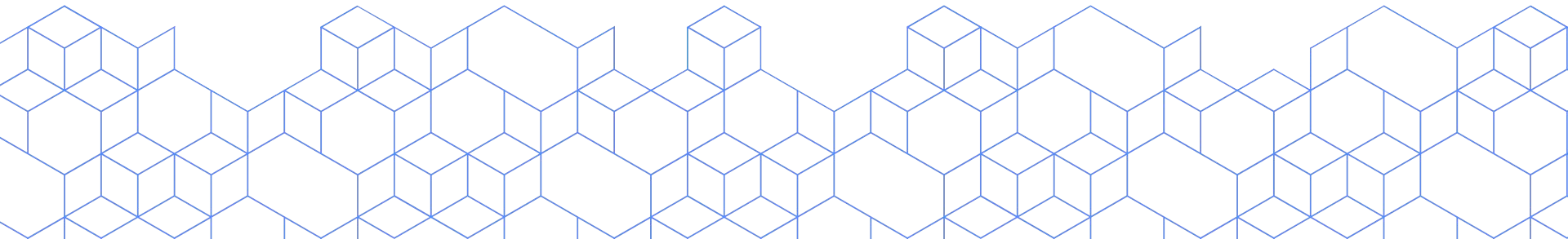
# Mergesort

6 5 3 1 8 7 2 4



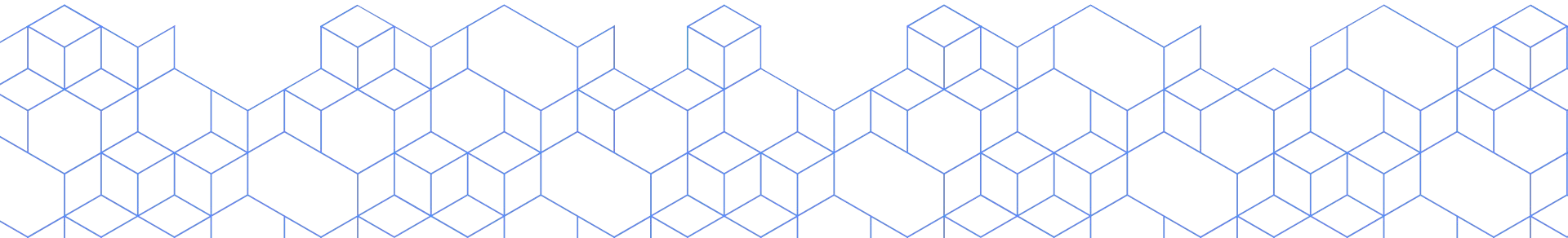
Criteria	Sorting algorithm
Only a few items	<b>Insertion Sort</b>
Items are mostly sorted already	<b>Insertion Sort</b>
Concerned about worst-case scenarios	<b>Heap Sort</b>
Interested in a good average-case behavior	<b>Quicksort</b>
Items are drawn from a uniform dense universe	<b>Bucket Sort</b>
Desire to write as little code as possible	<b>Insertion Sort</b>
Require stable sort	<b>Merge Sort</b>

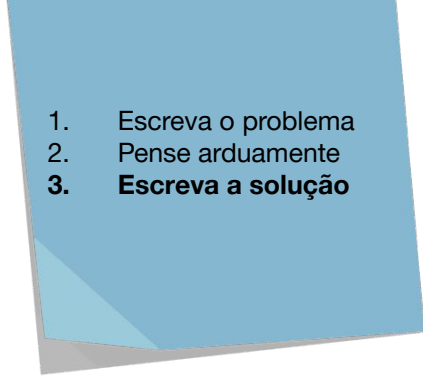
[Algorithms in a Nutshell: A Practical Guide](#)





E se fosse possível combinar os algoritmos  
de acordo com a necessidade?



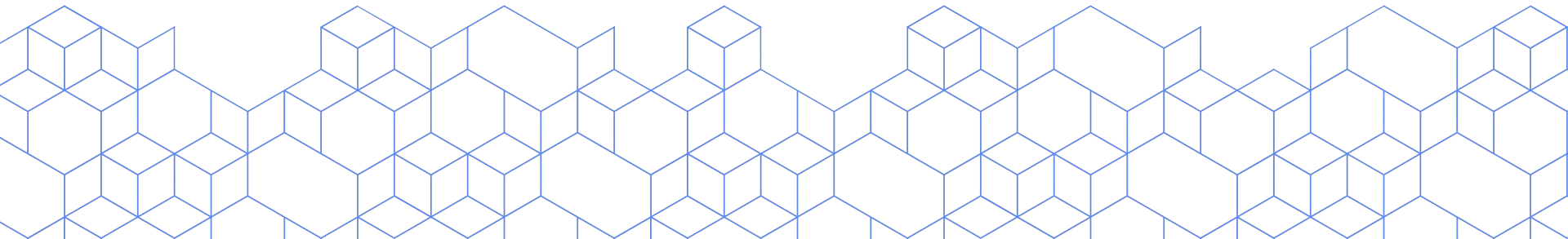
- 
1. Escreva o problema
  2. Pense arduamente
  3. **Escreva a solução**

# Solução :

## Timsort

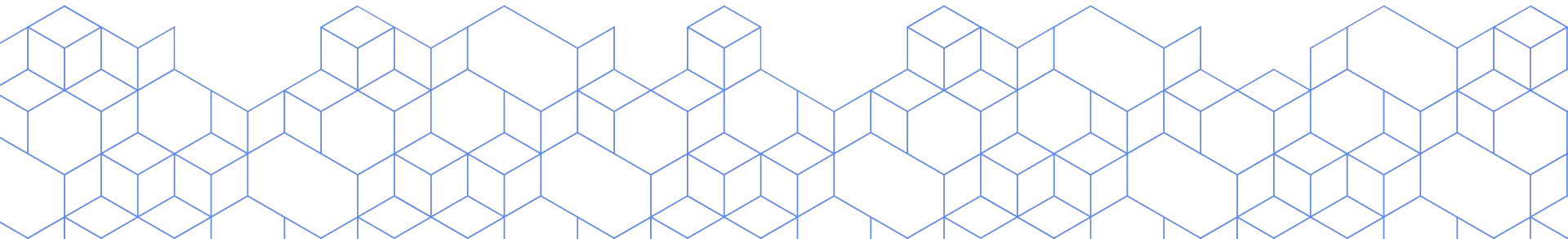
# Timsort

- Eficiente para dados do mundo real
- Usado em **Python**, [Java](#), plataforma **Android** e **GNU Octave**
- Escolhe uma abordagem com base na análise da lista.
- Rápido,  $O(n \log n)$ , estável



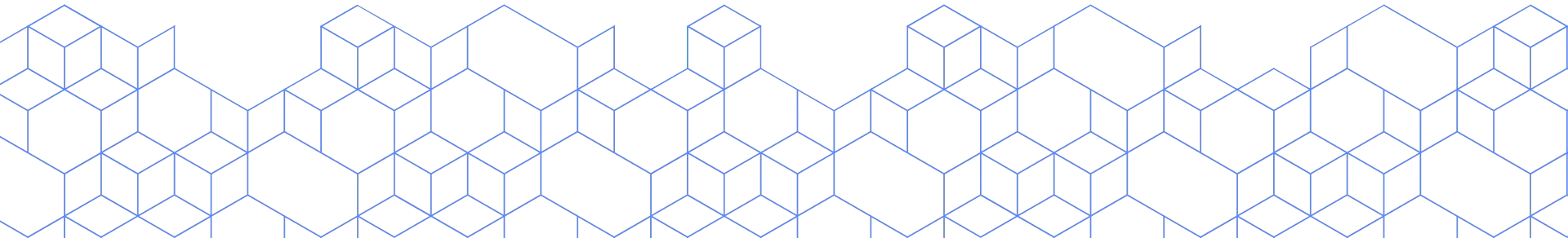
Se **N**  $\leq$  **64**, Timsort usa o método de ordenação por inserção para ordenar os elementos e não usa sofisticacões

[Python Docs](#)

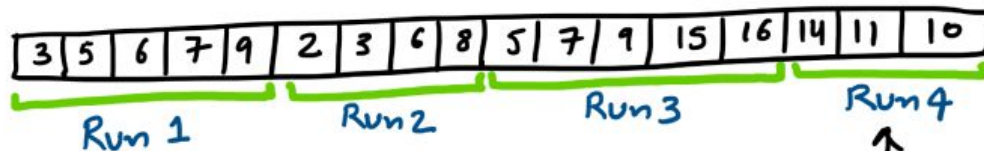


Se  $N > 64$

1. Cria **sequências** de um tamanho fixo (“run”)
2. **Ordena** essas sequências separadamente
3. **Mergea** as sequências de acordo com uma regra
4. **Otimiza o merge** usando o modo galope



Procura seqüências que aumentam ou diminuem,  
invertendo as seqüências que diminuem



↑  
decreasing order  
swap elements in-place

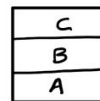
10 11 14  
Run 4

Usa uma **pilha** para controlar os **três itens mais recentes** e cria **duas leis** para decidir se a execução atual deve ser mergeada com as execuções anteriores

1.  $A > B + C$

2.  $B > C$

A:30 B:20 C:10

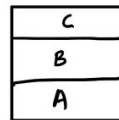


Merge B  
and C  
→

A:30 Bc:30



A:400 B:500 C:1000



Merge  
A and B  
→

AB: 900 C:1000

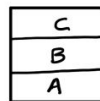


Usa uma **pilha** para controlar os **três itens mais recentes** e cria **duas leis** para decidir se a execução atual deve ser **mergeada** com as execuções anteriores

executar o  
mergesort

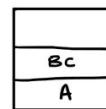
1.  $A > B + C$
2.  $B > C$

A:30 B:20 C:10

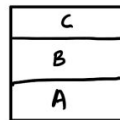


Merge B  
and C

A:30 Bc:30



A:400 B:500 C:1000



Merge  
A and B

AB: 900 C:1000

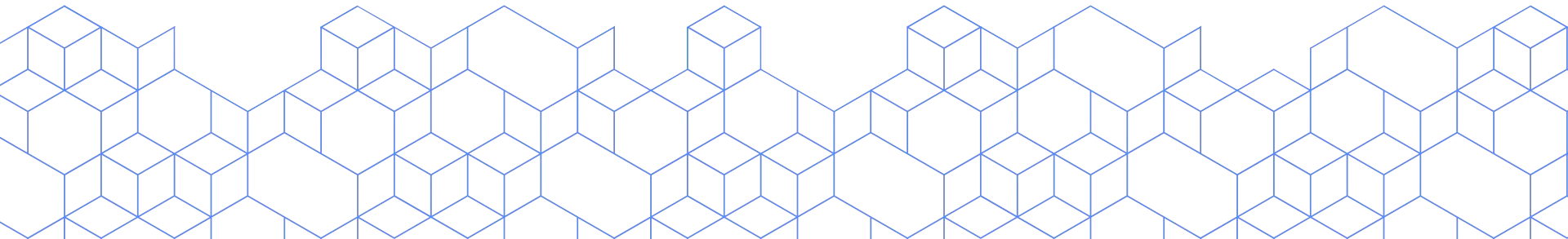




# Otimização com o modo de Galope

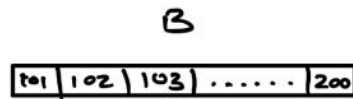
No processo de merge o algoritmo assume que,  
se **muitos valores da execução A forem inferiores aos valores da execução B**, é provável que A continue a ter valores menores que B.

[Python Docs](#)



# Otimização com o modo de Galope

Timsort realiza uma **busca binária** pela posição apropriada de B[0] em A[0].



1 < 101

2 < 101

3 < 101

⋮

7 < 101

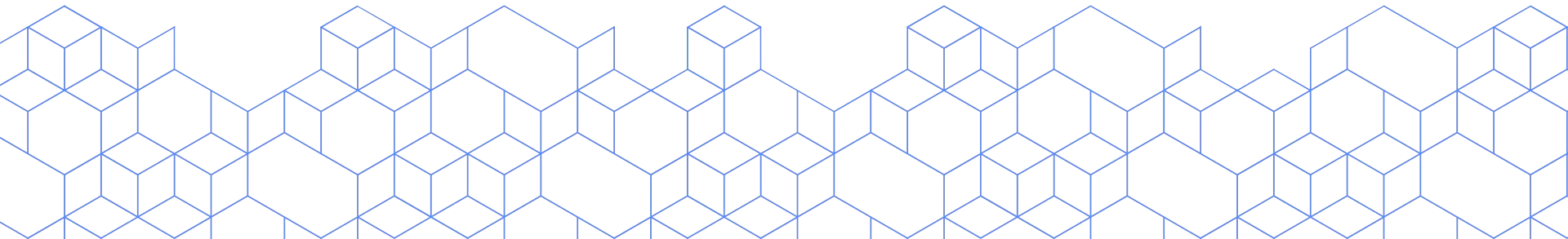
(Gallop Mode ON)

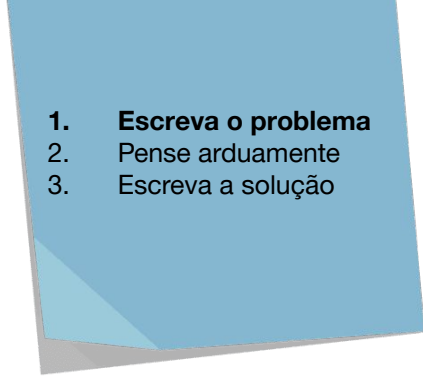
Binary Search for  
101 inside  
array A.



```
In [1]: import random
...: import timeit
...:
...: unsorted_array = []
...: unsorted_array = [random.randint(0, 1000000) for i in range(0, 1000000)]
...: unsorted_array.sort()
...: unsorted_array[:20]
```

```
Out[1]: [1, 3, 5, 6, 6, 7, 8, 9, 9, 9, 9, 10, 11, 11, 14, 16, 16, 18, 20, 20]
```

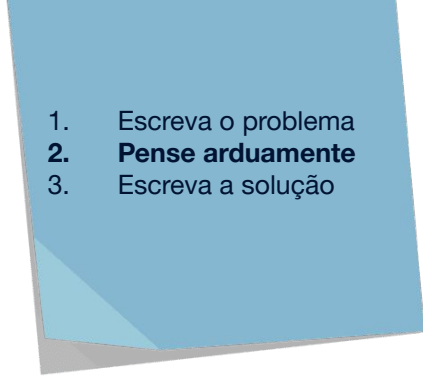


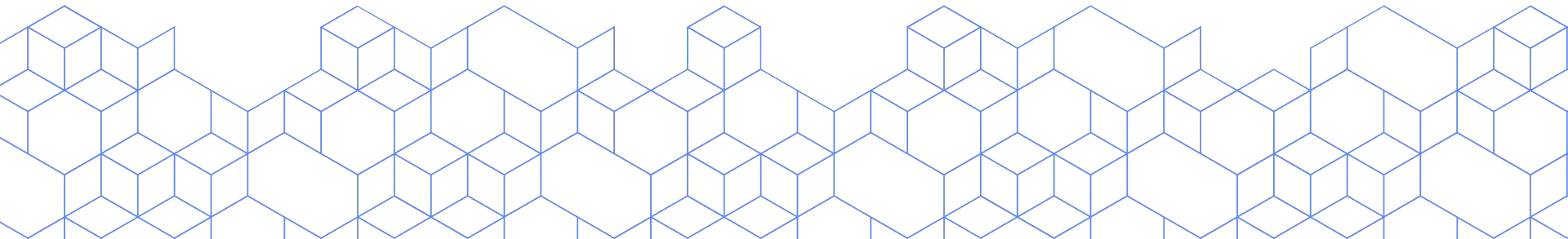
- 
1. **Escreva o problema**
  2. Pense arduamente
  3. Escreva a solução

**Problema :**  
Como buscar itens?

# Existem várias opções...

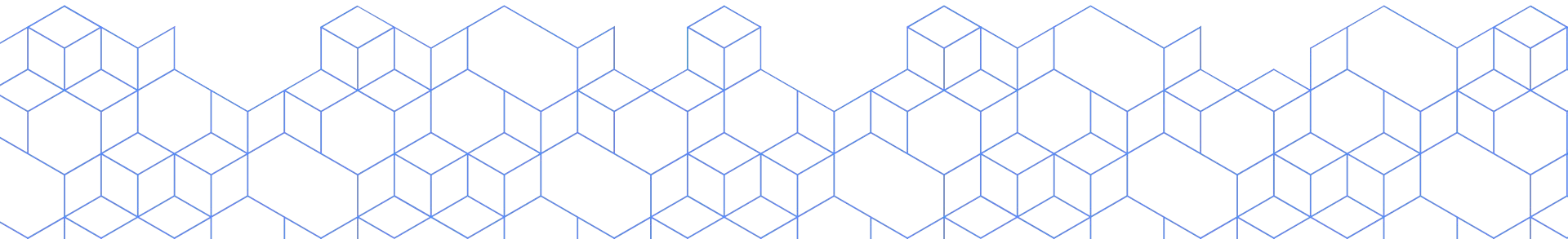
- Base de Dados
  - SQL
- Listas. Matrizes
  - Busca Binária
- Tabelas Hash
  - Buscas Baseadas em Hash


- 
1. Escreva o problema
  2. **Pense arduamente**
  3. Escreva a solução



# Tabela Hash

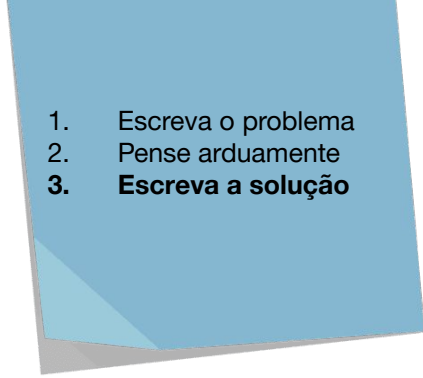
- Alternativa para pesquisar coleções maiores e **não ordenadas**
- **Função hash** para transformar uma **chave em um índice** em uma tabela de hash
- $\text{hash}(\text{"chave"}) == \text{hash}(\text{"chave"})$
- $\text{hash}(\text{"outra chave"}) \neq \text{hash}(\text{"chave"})$





```
In [1]: def get_item(item):
...:     return abs(hash(item) % 4)
...:
...: # Inserir a lista de lugares da python brasil
...: itens_list = [None] * 4
...:
...: # Inserir ribeirão preto
...: index = get_item('pybr19')           # list index 3
...: itens_list[index] = 'ribeirão preto'
...: # Inserir natal
...: index = get_item('pybr18')           # list index 0
...: itens_list[index] = 'natal'
...: itens_list
Out[1]: ['natal', None, None, 'ribeirão preto']

In [2]: index = get_item('pybr18')
...: itens_list[index]
Out[2]: 'natal'
```


- 
1. Escreva o problema
  2. Pense arduamente
  3. **Escreva a solução**

# Solução : Dicionários





**Python é construído em  
torno de dicionários**



```
from __future__ import division
import sys
```

```
class PythonBrasil:
```

```
    def __init__(self, v0, v1, v2, v3, v4):
```

```
        self.pybr19 = v0
```

```
        self.pybr18 = v1
```

```
        self.pybr17 = v2
```

```
        self.pybr16 = v3
```

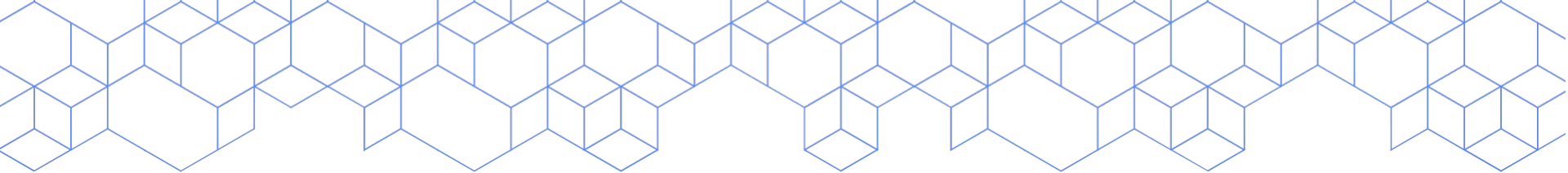
```
        self.pybr15 = v4
```

```
    def __repr__(self):
```

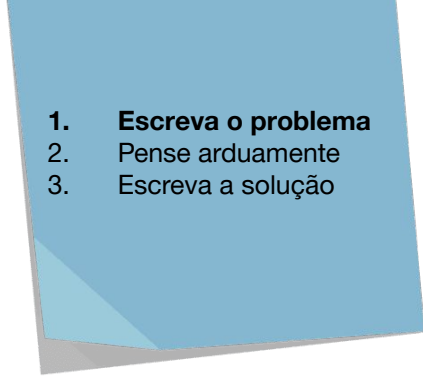
```
        return 'PythonBrasil(%r, %r, %r, %r, %r)' % (
```

```
            self.pybr19, self.pybr18, self.pybr17, self.pybr16, self.pybr15
```

```
        )
```



```
places = PythonBrasil('ribeirão preto', 'natal', 'belo horizonte', 'florianopolis', 'sao jose dos campos')
vars(places)
# {'pybr19': 'ribeirão preto',
#  'pybr18': 'natal',
#  'pybr17': 'belo horizonte',
#  'pybr16': 'florianopolis',
#  'pybr15': 'sao jose dos campos'}
```

- 
1. **Escreva o problema**
  2. Pense arduamente
  3. Escreva a solução

# Problema :

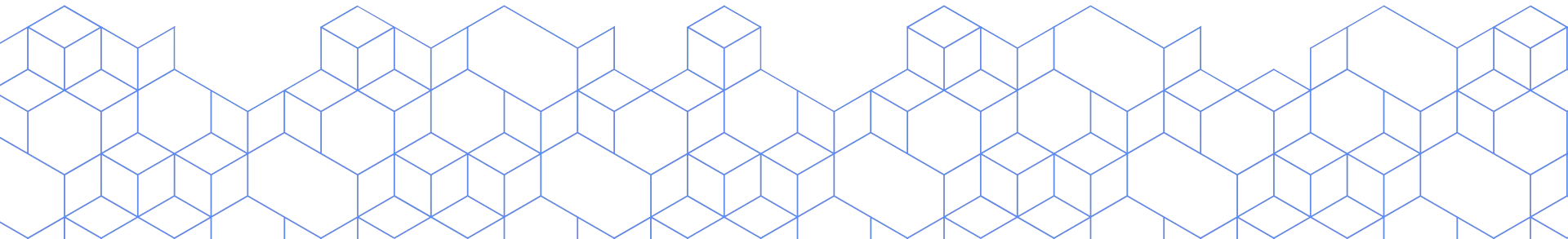
Como dicionários funcionam?

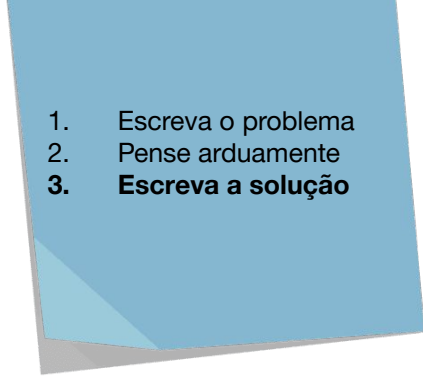
# Tabela Hash

1. Escreva o problema
2. **Pense arduamente**
3. Escreva a solução

- Estrutura de dados usada por dicionários
- Basicamente uma lista
- Hash das chaves é necessário para revelar os valores

```
[['pybr19', 'ribeirão preto'], ['pybr18', 'natal']]
```



- 
1. Escreva o problema
  2. Pense arduamente
  3. **Escreva a solução**

# Solução:

## Dicionários na prática

# Configuração

```
keys = [  
    'pybr19', 'pybr18', 'pybr17', 'pybr16', 'pybr15',  
    'pybr14', 'pybr13'  
]  
values = [  
    'ribeirão preto', 'natal', 'belo horizonte',  
    'florianopolis', 'sao jose dos campos', 'recife',  
    'brasilia'  
]  
hashes = list(map(abs, map(hash, keys)))  
entries = list(zip(hashes, keys, values))  
  
[(6519378555130876693, 'pybr19', 'ribeirão preto'),  
 (1831110896825541078, 'pybr18', 'natal'),  
 (9167591958126575224, 'pybr17', 'belo horizonte'),  
 (4819543372031726241, 'pybr16', 'florianopolis'),  
 (5067670214198873854, 'pybr15', 'sao jose dos campos'),  
 (2940889712379195968, 'pybr14', 'recife'),  
 (8949678210916869228, 'pybr13', 'brasilia')]
```




```
In [2]: d = dict()
...: d['pybr19'] = 'ribeirão preto'
...: bits(hash('pybr19'))[-3:]
...:
Out[2]: '101'
"""
```

-----			
Idx	hash	key	value
-----			
000			
001			
010			
011			
100			
101			
110			
111			

```
"""
```





```
In [2]: d = dict()
...: d['pybr19'] = 'ribeirão preto'
...: bits(hash('pybr19'))[-3:]      # últimos 3 bits
...:
```

```
Out[2]: '101'
```

```
"""
```

-----				
	Idx	hash	key	value
-----				
	000			
	001			
	010			
	011			
	100			
	101	_00010101	pybr19	ribeirão preto
	110			
	111			

```
"""
```



```
In [3]: d['pybr18'] = 'natal'
...: bits(hash('pybr18'))[-3:]
...:
```

```
Out[3]: '110'
"""
```

-----				
Idx	hash	key		value
-----				
000				
001				
010				
011				
100				
101	_00010101	pybr19	ribeirão preto	
110	_11010110	pybr18	natal	
111				

```
-----
"""
```



```
In [4]: d['pybr17'] = 'belo horizonte'
...: bits(hash('pybr17'))[-3:]
...:
Out[4]: '000'
"""
```

-----				
	Idx	hash	key	value
-----				
	000	_01111000	pybr17	belo horizonte
	001			
	010			
	011			
	100			
	101	_00010101	pybr19	ribeirão preto
	110	_11010110	pybr18	natal
	111			

```
-----
"""
```

In [5]: d['pybr16'] = 'florianopolis'  
...: bits(hash('pybr16'))[-3:]  
...:

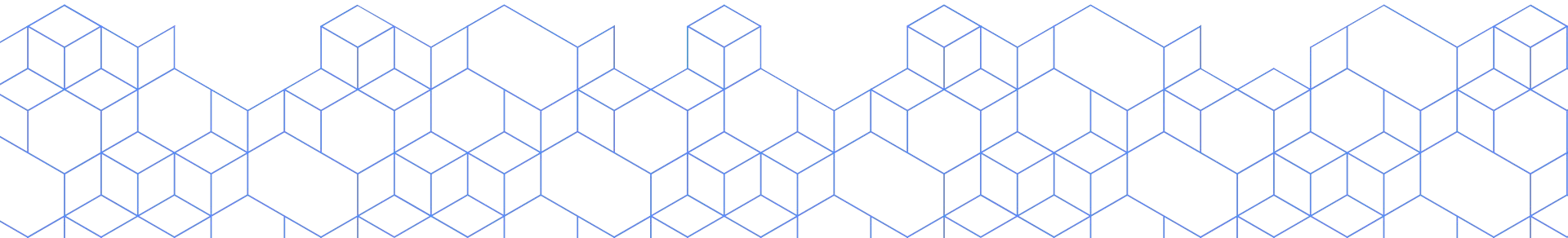
Out[5]: '001'


"""

-----				
Idx		hash	key	value
-----				
000		_01111000	pybr17	belo horizonte
001		_10100001	pybr16	florianopolis
010				
011				
100				
101		_00010101	pybr19	ribeirão preto
110		_11010110	pybr18	natal
111				

"""

1. Calcular o hash
2. Selecionar uma parte
3. Procurar no espaço correspondente





```
In [6]: d['pybr15'] = 'sao jose dos campos'
...: bits(hash('pybr15'))[-3:]
...:
```

```
Out[6]: '110'
"""
```

-----				
Idx		hash	key	value
-----				
000		_01111000	pybr17	belo horizonte
001		_10100001	pybr16	florianopolis
010				
011				
100				
101		_00010101	pybr19	ribeirão preto
110		_11010110	pybr18	natal
111				

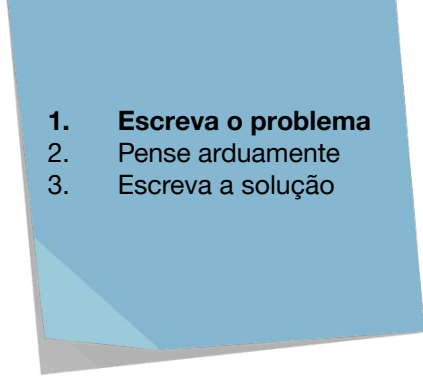
```
"""
```

In [6]: d['pybr15'] = 'sao jose dos campos'  
...: bits(hash('pybr15'))[-3:]  
...:

Out[6]: '110'  
"""

-----				
Idx		hash	key	value
-----				
000		_01111000	pybr17	belo horizonte
001		_10100001	pybr16	florianopolis
010				
011				
100				
101		_00010101	pybr19	ribeirão preto
110		_11010110	pybr18	natal
111				

-----  
"""

- 
1. **Escreva o problema**
  2. Pense arduamente
  3. Escreva a solução

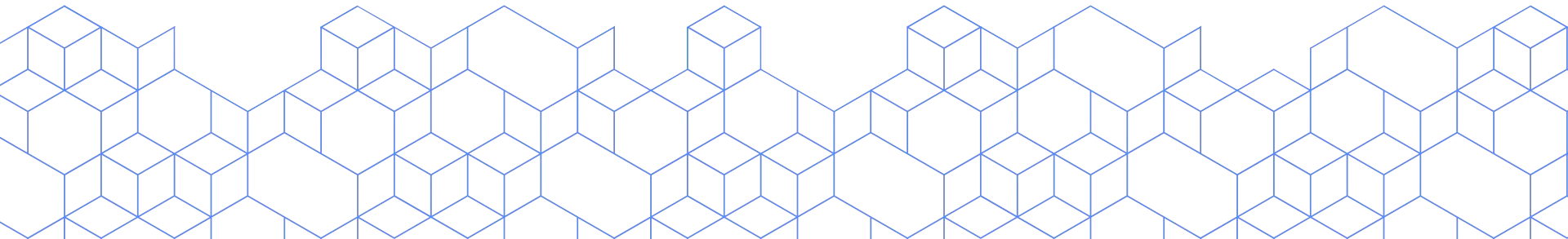
**Problema :**  
Como resolver colisões?

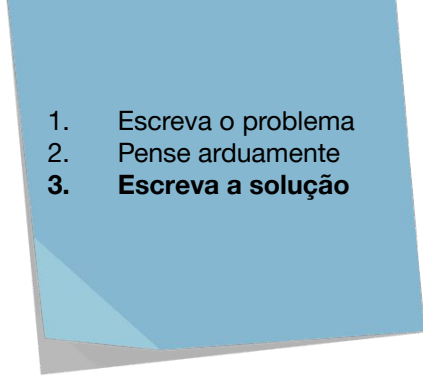


1. Escreva o problema
2. **Pense arduamente**
3. Escreva a solução

# Colisão

Quando duas chaves no dicionário  
possuem a mesma terminação



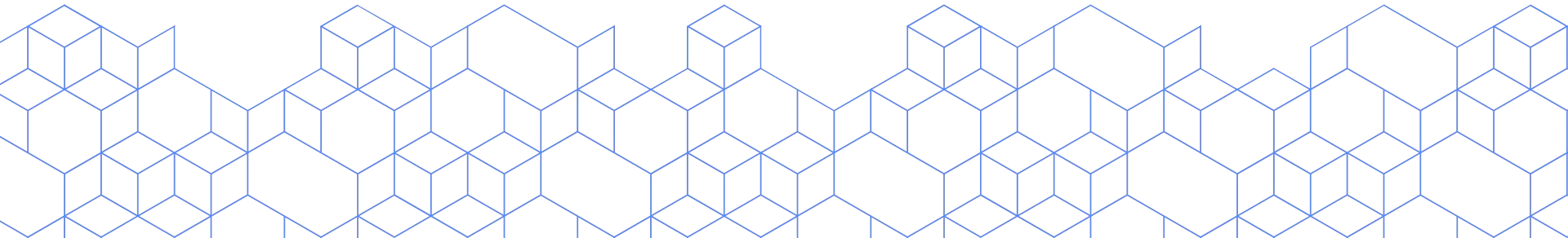
- 
1. Escreva o problema
  2. Pense arduamente
  3. **Escreva a solução**

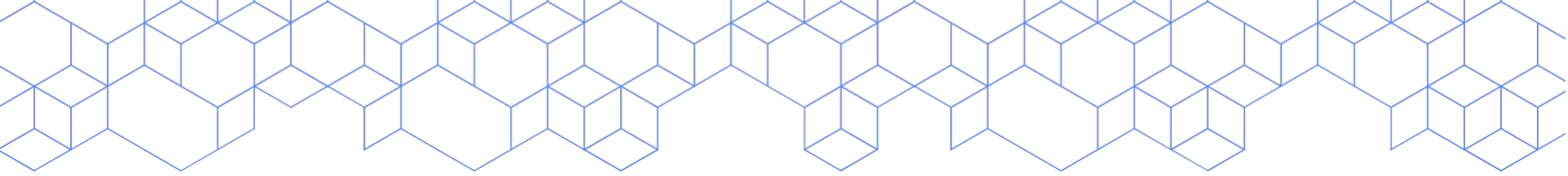
# Solução:

Endereçamento Aberto com Múltiplos Hashes

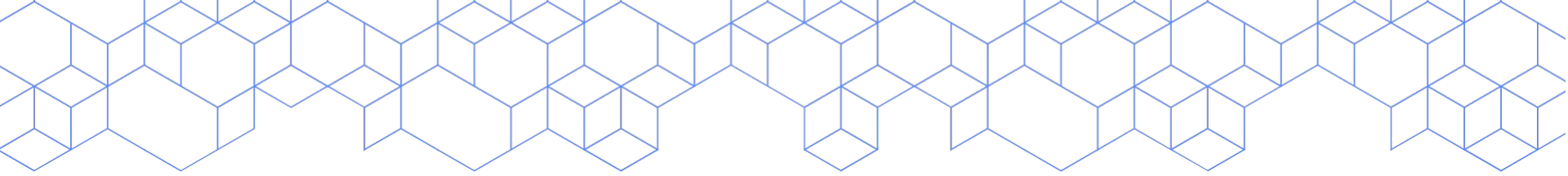
## Endereçamento Aberto

- Tornar a tabela mais densa.
- Lida com colisões procurando linearmente na tabela até encontrar um registro vazio ou o registro buscado.



- 
- Usar todos os bits no hash
  - Usar um gerador de números
  - Python 1.5.2

```
def open_addressing_multihash(n, entries):  
    table = [None] * n  
    for h, key, value in entries:  
        perturb = h  
        i = h % n  
        while table[i] is not None:  
            print('%r collided with %r' % (key, table[i][0]))  
            i = (5 * i + perturb + 1) % n  
            perturb  $\gg$  5  
        table[i] = (key, value)  
    print(table)
```

- 
- Usar todos os bits no hash
  - Usar um gerador de números
  - Python 1.5.2

```
def open_addressing_multihash(n, entries):  
    table = [None] * n  
    for h, key, value in entries:  
        perturb = h  
        i = h % n  
        while table[i] is not None:  
            print('%r collided with %r' % (key, table[i][0]))  
            i = (5 * i + perturb + 1) % n  
            perturb  $\gg$  5  
        table[i] = (key, value)  
    print(table)
```

```
In [6]: d['pybr15'] = 'sao jose dos campos'
...: bits(hash('pybr15'))[-3:]
...:
```

```
Out[6]: '110' # pybr15 colidiu com pybr18
"""
```

-----				
Idx		hash	key	value
-----				
000		_01111000	pybr17	belo horizonte
001		_10100001	pybr16	florianopolis
010				
011				
100				
101		_00010101	pybr19	ribeirão preto
110		_11010110	pybr18	natal
111				
-----				

```
"""
```



```
In [6]: open_addressing_multihash(8, entries[:5])
```

```
'pybr15' collided with 'pybr18'
```

```
'pybr15' collided with 'pybr19'
```

```
'pybr15' collided with 'pybr16'
```

```
'pybr15' collided with 'pybr16'
```

```
Out[6]:
```

```
[('000', 'pybr17', 'belo horizonte'),
```

```
 ('001', 'pybr16', 'florianopolis'),
```

```
 None,
```

```
 ('110', 'pybr15', 'sao jose dos campos'),
```

```
 None,
```

```
 ('101', 'pybr19', 'ribeirão preto'),
```

```
 ('110', 'pybr18', 'natal'),
```

```
 None]
```

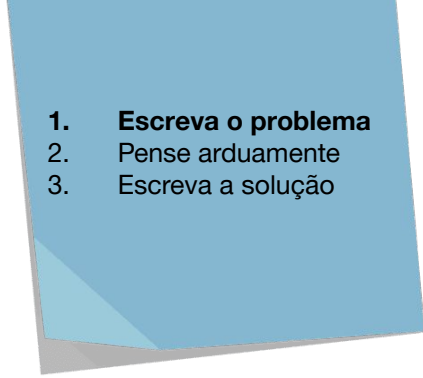
```
In [6]: d['pybr15'] = 'sao jose dos campos'
...: bits(hash('pybr15'))[-3:]
...:
Out[6]: '110'
```

```
"""
# pybr15 foi salvo em 011
# dicionário % cheio
```

-----				
Idx		hash	key	value
-----				
000		_01111000	pybr17	belo horizonte
001		_10100001	pybr16	florianopolis
010				
011		_11111110	pybr15	sao jose dos campos
100				
101		_00010101	pybr19	ribeirão preto
110		_11010110	pybr18	natal
111				
-----				

```
"""
```



- 
1. **Escreva o problema**
  2. Pense arduamente
  3. Escreva a solução

# Problema :

Como definir o tamanho do dicionário?

# Detalhes de implementação

- PyDict\_MINSIZE (8)
- 8 permite dicionários com não mais que 5 entradas

1. Escreva o problema
2. **Pense arduamente**
3. Escreva a solução



```
In [2]: d = dict()
...: d['pybr19'] = 'ribeirão preto'
...: bits(hash('pybr19'))[-3:]
...:
Out[2]: '101'
```

```
"""
```

-----			
Idx	hash	key	value
-----			
000			
010			
001			
011			
100			
101			
110			
111			
-----			

```
"""
```

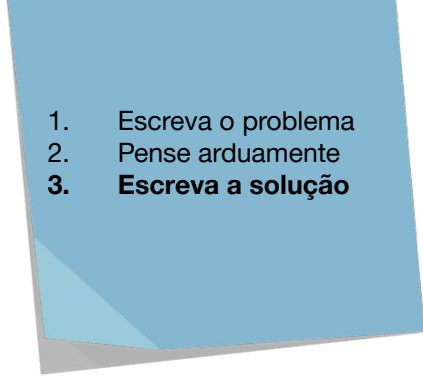


```
In [1]:  
...: import sys  
...: d = dict()  
...: sys.getsizeof(d)
```

```
Out[1]: 240
```

```
In [2]: d['pybr19'] = 'ribeirão preto'  
...: d['pybr18'] = 'natal'  
...: d['pybr17'] = 'belo horizonte'  
...: d['pybr16'] = 'florianopolis'  
...: d['pybr15'] = 'sao jose dos campos'  
...: sys.getsizeof(d)
```

```
Out[2]: 240
```

- 
1. Escreva o problema
  2. Pense arduamente
  3. **Escreva a solução**

# Solução :

## Redimensionamento dinâmico

- Redimensionamento acontece quando o dicionário está  $\frac{2}{3}$  **cheio**
- **< 50k** itens, tamanho \* 4
- **> 50k** itens, tamanho \* 2



```
In [1]:  
...: import sys  
...: d = dict()  
...: sys.getsizeof(d)
```

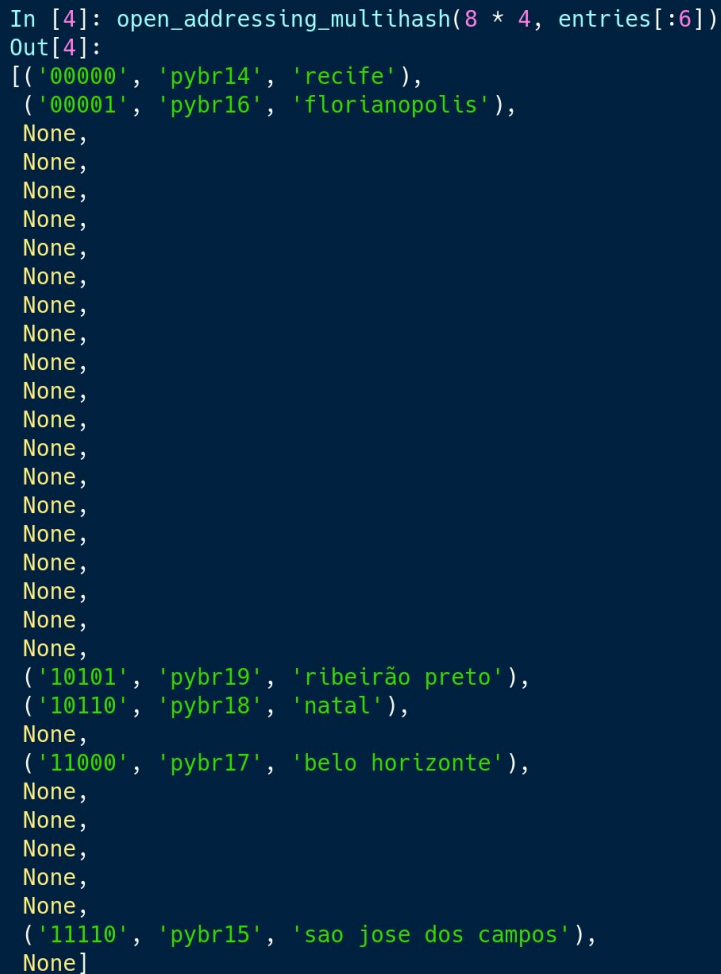
```
Out[1]: 240
```

```
In [2]: d['pybr19'] = 'ribeirão preto'  
...: d['pybr18'] = 'natal'  
...: d['pybr17'] = 'belo horizonte'  
...: d['pybr16'] = 'florianopolis'  
...: d['pybr15'] = 'sao jose dos campos'  
...: sys.getsizeof(d)
```

```
Out[2]: 240
```

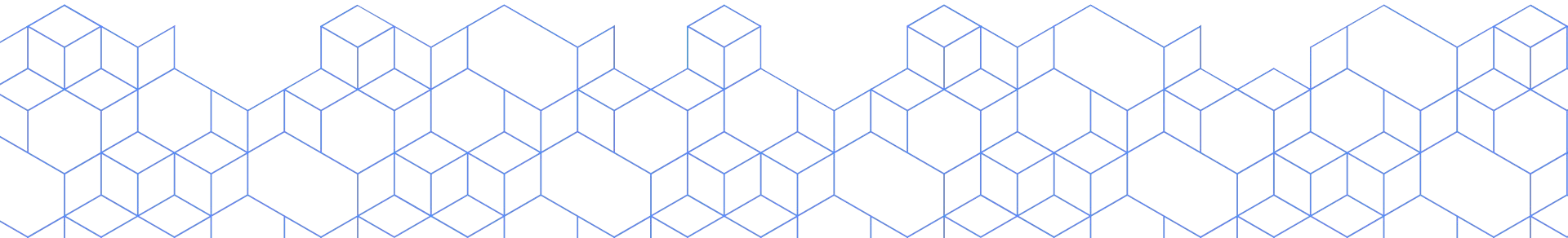
```
In [3]: d['pybr14'] = 'recife'  
...: sys.getsizeof(d)
```

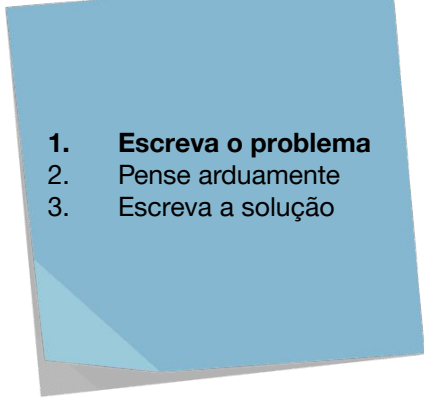
```
Out[3]: 368
```





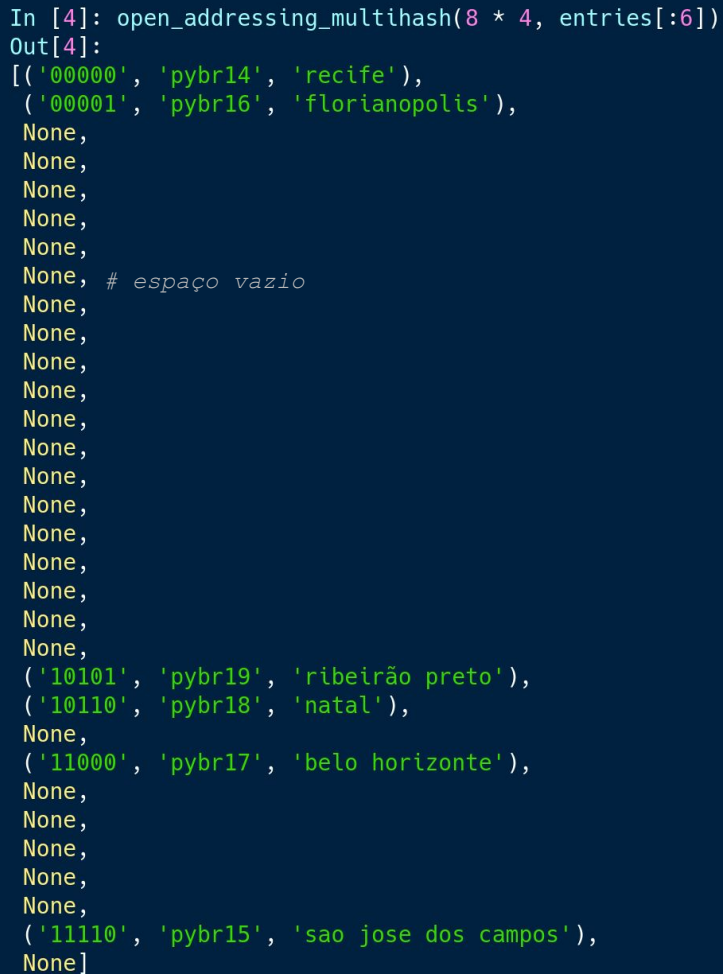
1. **Enche gradualmente** a medida que itens são adicionados
2. Se torna subitamente menos cheio à medida que o dicionário é **redimensionado**
3. Gerando um **desempenho médio excelente**

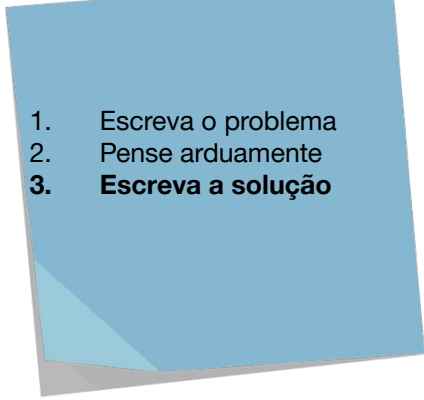


- 
1. **Escreva o problema**
  2. Pense arduamente
  3. Escreva a solução

# Problema :

Como economizar memória no dicionário?



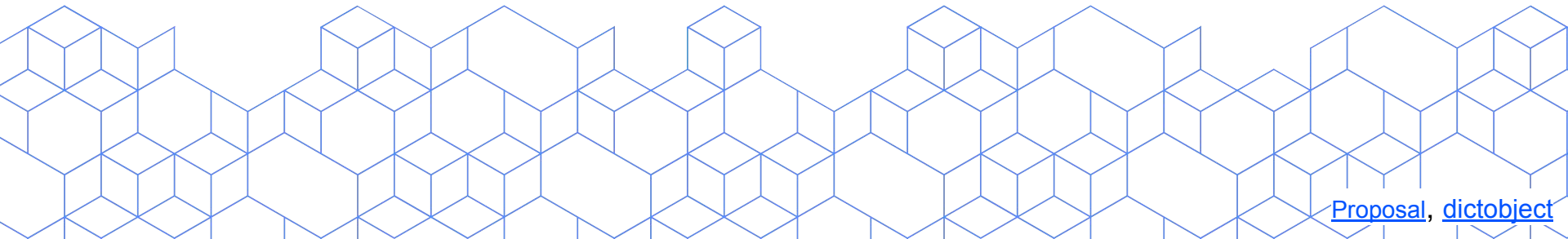
- 
1. Escreva o problema
  2. Pense arduamente
  3. **Escreva a solução**

# Solução

## Dicionários Compactos


# Dicionários Compactos

- Economizar **memória**
- Entradas devem ser armazenadas em um **tabela densa** referenciada por uma **tabela esparsa** de **índices**.
- Somente o **layout** dos dados é alterado.
- A tabela hash e os algoritmos de otimização permanecem os mesmos





```
def compact_and_ordered(n, entries):  
    import pprint  
    table = [None] * n  
    for pos, entry in enumerate(entries):  
        h = perturb = entry[0]  
        i = h % n  
        while table[i] is not None:  
            i = (5 * i + perturb + 1) % n  
            perturb >>= 5  
        table[i] = pos  
    pprint.pprint(entries)  
    return table
```



```
In [53]: compact_and_ordered(8, entries[:5])
[(6519378555130876693, 'pybr19', 'ribeirão preto'),
 (1831110896825541078, 'pybr18', 'natal'),
 (9167591958126575224, 'pybr17', 'belo horizonte'),
 (4819543372031726241, 'pybr16', 'florianopolis'),
 (5067670214198873854, 'pybr15', 'sao jose dos campos')]
Out[53]: [2, 3, None, 4, None, 0, 1, None]
```

In [53]: compact\_and\_ordered(8, entries[:5])  
[(6519378555130876693, 'pybr19', 'ribeirão preto'),  
(1831110896825541078, 'pybr18', 'natal'),  
(9167591958126575224, 'pybr17', 'belo horizonte'),  
(4819543372031726241, 'pybr16', 'florianopolis'),  
(5067670214198873854, 'pybr15', 'sao jose dos campos')]  
Out[53]: [2, 3, None, 4, None, 0, 1, None]

Idx	hash	key	value
000	_01111000	pybr17	belo horizonte
001	_10100001	pybr16	florianopolis
010			
011	_11111110	pybr15	sao jose dos campos
100			
101	_00010101	pybr19	ribeirão preto
110	_11010110	pybr18	natal
111			

"""



**sem\_dicionários\_compactos**  $(24+t) =$

$$24*8 = \mathbf{192}$$

**com\_dicionários\_compactos**  $(24*n + \mathbf{sizeof}(\text{index}) * t) =$

$$24*5 + 1*8 = \mathbf{128}$$

# Pegadinha #1

Porque a função hash retorna resultados diferentes entre sessões?

```
keys = [  
    'pybr19', 'pybr18', 'pybr17', 'pybr16', 'pybr15',  
    'pybr14', 'pybr13'  
]  
values = [  
    'ribeirão preto', 'natal', 'belo horizonte',  
    'florianopolis', 'sao jose dos campos', 'recife',  
    'brasilia'  
]  
hashes = list(map(abs, map(hash, keys)))  
entries = list(zip(hashes, keys, values))  
  
[(6519378555130876693, 'pybr19', 'ribeirão preto'),  
 (1831110896825541078, 'pybr18', 'natal'),  
 (9167591958126575224, 'pybr17', 'belo horizonte'),  
 (4819543372031726241, 'pybr16', 'florianopolis'),  
 (5067670214198873854, 'pybr15', 'sao jose dos campos'),  
 (2940889712379195968, 'pybr14', 'recife'),  
 (8949678210916869228, 'pybr13', 'brasilia')]
```



```
Python 3.6.8 (default, Oct 7 2019, 12:59:55)  
Type 'copyright', 'credits' or 'license' for more information  
IPython 7.8.0 -- An enhanced Interactive Python. Type '?' for help.
```

```
In [1]: hash('pybr19')  
Out[1]: 6280554434842480070
```



```
Python 3.6.8 (default, Oct 7 2019, 12:59:55)  
Type 'copyright', 'credits' or 'license' for more information  
IPython 7.8.0 -- An enhanced Interactive Python. Type '?' for help.
```

```
In [1]: hash('pybr19')  
Out[1]: -5116335369826839940
```



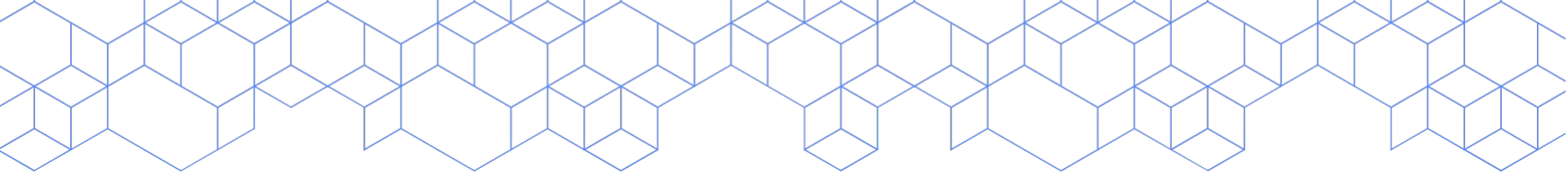
```
Python 3.6.8 (default, Oct 7 2019, 12:59:55)  
Type 'copyright', 'credits' or 'license' for more information  
IPython 7.8.0 -- An enhanced Interactive Python. Type '?' for help.
```

```
In [1]: hash('pybr19')  
Out[1]: 6280554434842480070
```



```
Python 3.6.8 (default, Oct 7 2019, 12:59:55)  
Type 'copyright', 'credits' or 'license' for more information  
IPython 7.8.0 -- An enhanced Interactive Python. Type '?' for help.
```

```
In [1]: hash('pybr19')  
Out[1]: -5116335369826839940
```



“Os valores `__hash__()` dos objetos str e bytes são "misturados" com um **valor aleatório imprevisível**. Embora permaneçam constantes em um processo individual do Python, **não** são **previsíveis** entre **sessões repetidas do Python**.”

[https://docs.python.org/3/reference/datamodel.html#object.\\_\\_hash\\_\\_](https://docs.python.org/3/reference/datamodel.html#object.__hash__)



## Outros comportamentos interessantes

- Key-Sharing Dict
- Eficiência de **len()** e **pop()** ([aqui](#))
- Por que é mais lento iterar sobre uma **string pequena** do que sobre uma **lista pequena**? ([aqui](#))

# Referências

- Brandon Rhodes: The Mighty Dictionary (PyCon 2010) ([video](#))
- Modern Dictionaries by Raymond Hettinger ([video](#))
- <https://docs.python.org/3/reference/datamodel.html>
- <https://github.com/python/cpython/blob/b16e382c446d76ede22780b15c75f43c5f132e25/Objects/dictobject.c>
- <https://github.com/python/cpython/blob/b16e382c446d76ede22780b15c75f43c5f132e25/Objects/dictnotes.txt#L4>
- <https://github.com/python/cpython/tree/b16e382c446d76ede22780b15c75f43c5f132e25/Objects>
- [https://dev.to/s\\_awdesh/timsort-fastest-sorting-algorithm-for-real-world-problems--2jhd](https://dev.to/s_awdesh/timsort-fastest-sorting-algorithm-for-real-world-problems--2jhd)
- Grokking Algorithms: An illustrated guide for programmers and other curious people ([book](#))
- Timsort: The Fastest sorting algorithm for real-world problems. ([aqui](#))
- Timsort — the fastest sorting algorithm you've never heard of ([aqui](#))
- <https://github.com/python/cpython/blob/b16e382c446d76ede22780b15c75f43c5f132e25/Objects/listsort.txt>



# Obrigada!

Perguntas?

**Rebeca Sarai**

Software Developer

✉ [rebeca@vinta.com.br](mailto:rebeca@vinta.com.br)

🐦 [@\\_rebecasarai](https://twitter.com/_rebecasarai)

🐙 [/rsarai](https://github.com/rsarai)

**Feedbacks** são bem vindos:  
[rebeca@vinta.com.br](mailto:rebeca@vinta.com.br)

**Slides:** [bit.ly/pybr-talks](https://bit.ly/pybr-talks)

**Código:** [github.com/rsarai/talks](https://github.com/rsarai/talks)

