



```
[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](#)
- Torcedora do Náutico  

✉ rebeca@vinta.com.br

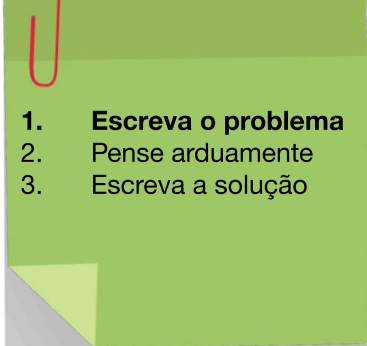
🐦 @_rebecasarai

🐙 /rsarai

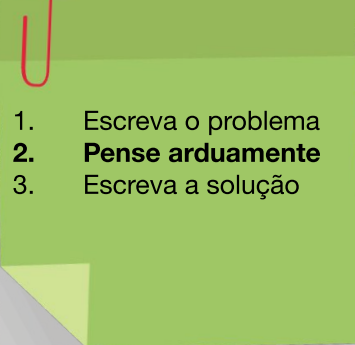
Slide de rob

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?

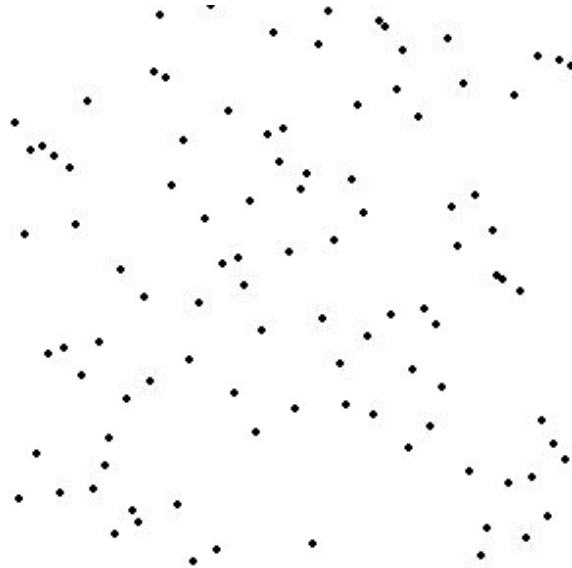


6 5 3 1 8 7 2 4

Ordenação por Inserção

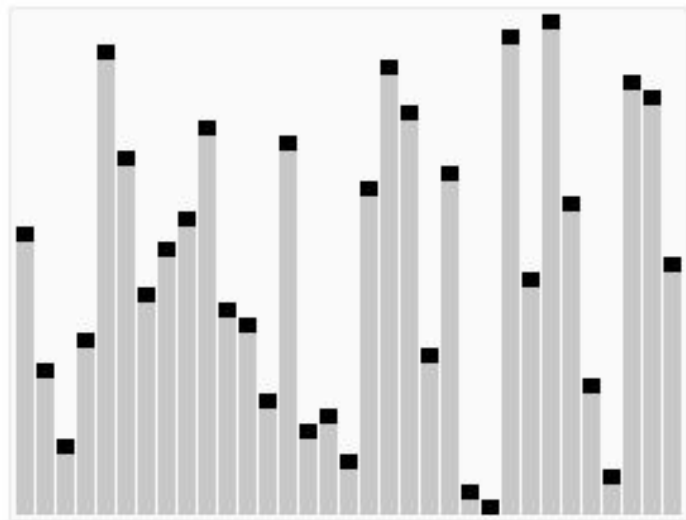
6 5 3 1 8 7 2 4

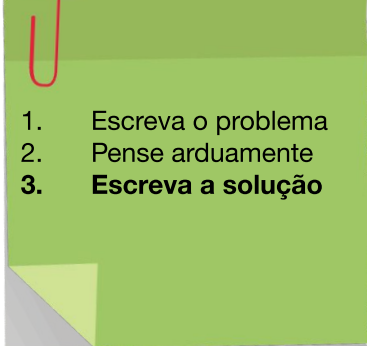
Ordenação por Seleção



**Dividir para
conquistar**

Quicksort



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

Solução : Timsort

<https://github.com/python/cpython/blob/b16e382c446d76ede22780b15c75f43c5f132e25/Objects/listsort.txt>

Timsort

- Rápido , $O(n \log n)$, estável
- Foi usado em **Python**, **Java**, plataforma **Android** e **GNU Octave**
- Escolhe uma abordagem com base na análise da lista.

<https://github.com/python/cpython/blob/b16e382c446d76ede22780b15c75f43c5f132e25/Objects/listsort.txt>

- If $N \leq 64$ then Tim sort uses binary insertion sort to sort the elements and doesn't go in fancy details.
-

Pegadinha :

Por que é mais rápido processar uma lista ordenada que uma lista desordenada?



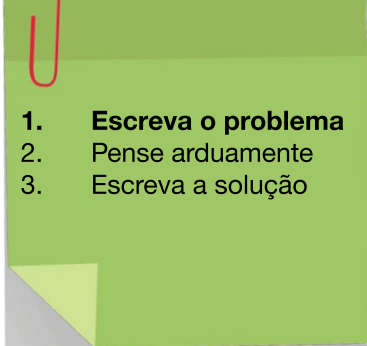
```
import time
import random

def process_array(array):
    start = time.time()
    processed_value = sum([i for i in array])
    end = time.time()
    print(end - start)

unsorted_array = [random.randint(0, 1000000) for i in range(0, 10000000)]
process_array(unsorted_array)
# 0.20371270179748535

unsorted_array.sort()
process_array(unsorted_array)
# 0.9198307991027832
```

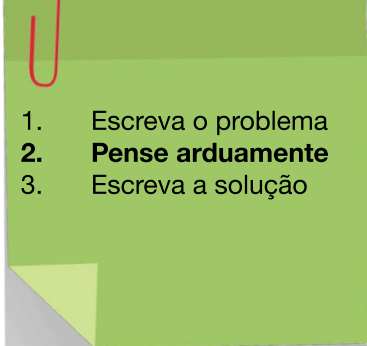

Branch Prediction

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

Problema :
Como encontrar itens?

Existem várias opções...

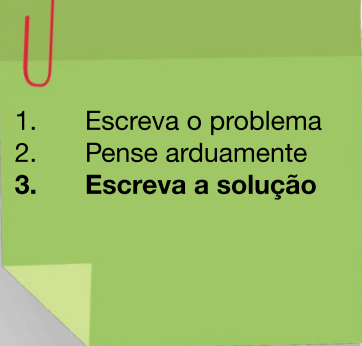
- Base de Dados.
 - SQL
- Listas. Matrizes.
 - Busca Binária.

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




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

Solução : Dicionários

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

**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):
```

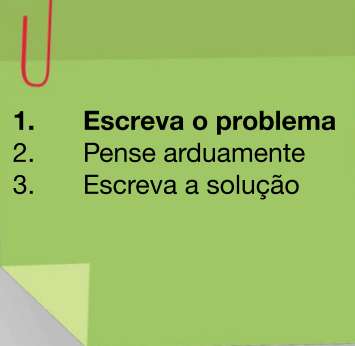
```
        return 'PythonBrasil(%r, %r, %r, %r, %r)' % (
```

```
            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'}
```

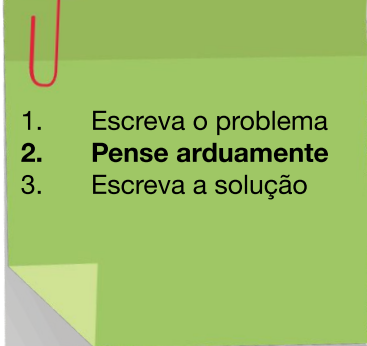



Problema :

Como dicionários funcionam?

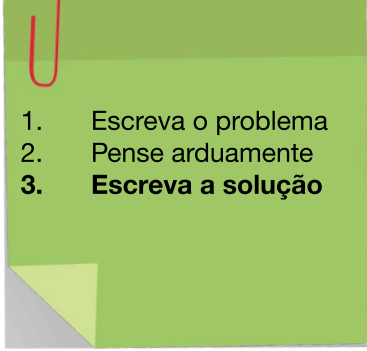
Tabela Hash

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

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


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

Solução : Dicionários

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

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:]

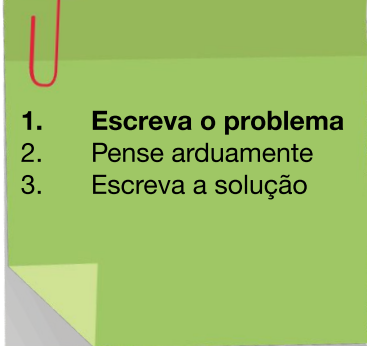
Out[6]: '110'

"""

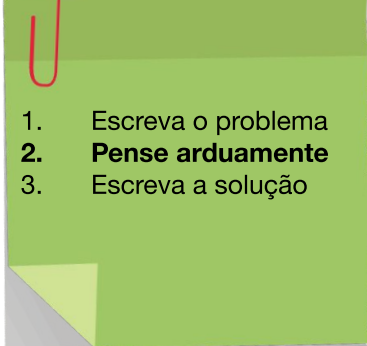
Idx		hash	key	value

000		_01111000		pybr17
001		_10100001		pybr16
010				
011				
100				
101		_00010101		pybr19
110		_11010110		pybr18
111				

"""

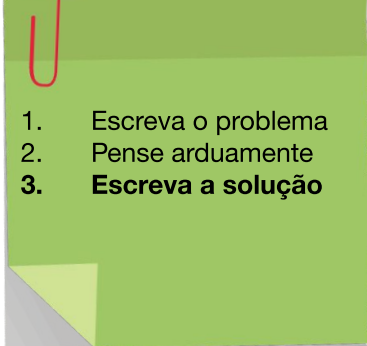
- 
1. **Escreva o problema**
 2. Pense arduamente
 3. Escreva a solução

Problema :
Como resolver colisões?

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

Colisão

Quando duas chaves no dicionário terminam da mesma forma

- 
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 >>= 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 >>= 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'),
```

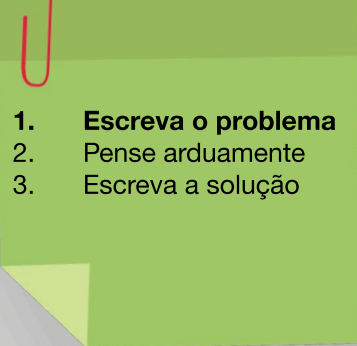
```
 None]
```

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			

```
"""
```

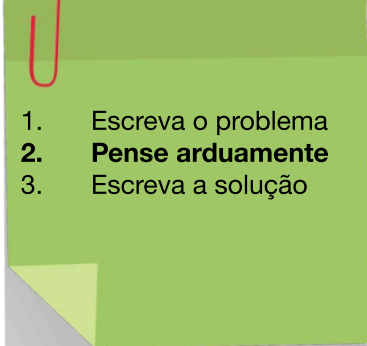


1. **Escreva o problema**
2. Pense arduamente
3. Escreva a solução

Problema :
Como definir o tamanho
do dicionário?

Detalhes de implementação

- PyDict_MINISIZE (8)
- 8 permite dicionários com não mais que 5 entradas

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

Detalh

- Py
- 8

```
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			

"""

<https://github.com/python/cpython/blob/b16e382c446d76ede22780b15c75f43c5f132e25/Objects/dictobject.c>




```
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)
```

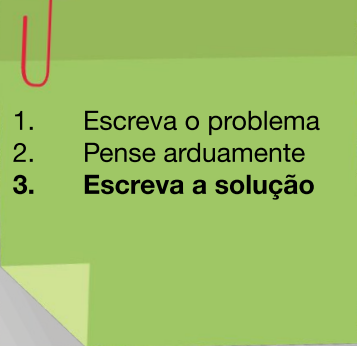
```
Out[2]: 240
```


In [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	_11111110	pybr15	sao jose dos campos
100			
101	_00010101	pybr19	ribeirão preto
110	_11010110	pybr18	natal
111			

"""



Solução :

Redimensionamento dinâmico

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



```
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)
```

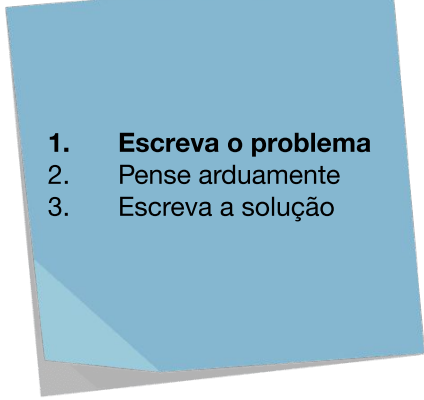
```
Out[3]: 368
```

<https://stackoverflow.com/questions/46026607/how-pythons-compact-dict-lookup-is-performed-with-int-value-inside-the-indicie>

```
In [4]: open_addressing_multihash(8 * 4, entries[:6])
Out[4]:
[(('00000', 'pybr14', 'recife'),
 ('00001', 'pybr16', 'florianopolis'),
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None),
 (('10101', 'pybr19', 'ribeirão preto'),
 ('10110', 'pybr18', 'natal'),
 None,
 ('11000', 'pybr17', 'belo horizonte'),
 None,
 None,
 None,
 None,
 None,
 None),
 (('11110', 'pybr15', 'sao jose dos campos'),
 None)]
```

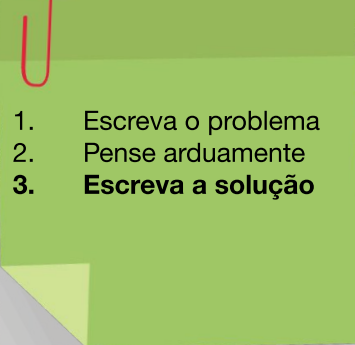
1. Enche gradualmente a medida que itens são adicionados
2. Se torna subitamente menos cheio à medida que o dicionário é redimensionado
3. Gerando um desempenho médio excelente

```
In [4]: open_addressing_multihash(8 * 4, entries[:6])
Out[4]:
[('00000', 'pybr14', 'recife'),
 ('00001', 'pybr16', 'florianopolis'),
 None,
 None,
 None,
 None,
 None,
 # espaço vazio
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 None,
 ('10101', 'pybr19', 'ribeirão preto'),
 ('10110', 'pybr18', 'natal'),
 None,
 ('11000', 'pybr17', 'belo horizonte'),
 None,
 None,
 None,
 None,
 None,
 ('11110', 'pybr15', 'sao jose dos campos'),
 None]
```


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

Problema :

Como economizar memória no dicionário?



Solução


Dicionários Compactos

Dicionários Compactos

- Economizar memória
- Entradas devem ser armazenadas em uma 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 $(24 * n + \text{sizeof}(\text{index}) * t) = 24 * 5 + 1 * 8 = \mathbf{128}$

Pegadinha :

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
```

“Por padrão, os valores `__hash__()` dos objetos `str` e `bytes` são "salgados" com um valor aleatório imprevisível. Embora permaneçam constantes em um processo individual do Python, não são previsíveis entre invocações repetidas do Python.”

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](#))

Obrigada!

Perguntas?

Rebeca Sarai

Software Developer

✉ rebeca@vinta.com.br

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

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

Feedbacks são bem vindos:
rebeca@vinta.com.br



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
- Grokking Algorithms: An illustrated guide for programmers and other curious people ([book](#))
- Timsort: The Fastest sorting algorithm for real-world problems. ([aqui](#))
- <https://github.com/python/cpython/blob/b16e382c446d76ede22780b15c75f43c5f132e25/Objects/listsort.txt>