

TRYBE

Modulo IV – Ciência da Computação

Bloco 38 - Estrutura de Dados: Hash e Set

1) Estrutura de dados II - Hash maps & Dict

O que é hashing em computação

Transformar um dado em uma **representação numérica única**. A saída é sempre a mesma quando tiver a mesma entrada.

3 tipos de hash functions

- De **valor** para valor;
- **Criptográficas**, via algoritmo pronto;
- **Checksum**, soma de verificação. *Ex de uso: verificar integridade de download.*

Hashmap: usando hashing para estruturar dados

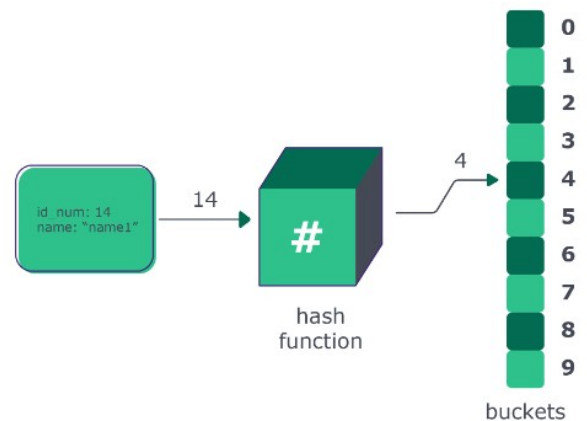
Estrutura de dados do tipo **chave-valor**.

A estratégia de armazenamento de dados da hashmap é
1/ submeter o dado a um procedimento matemático (hash function)...

2/...para obter um endereço único onde ela será guardada (**address**)...

3/...e armazenar o endereço no lugar certo (**bucket**)

Permite consultar e inserir em lista em **O(1)**: sem precisar percorrer array.



```
hashmap.py > ...
1 class Employee:
2     def __init__(self, id_num, name):
3         self.id_num = id_num
4         self.name = name
5
6 class HashMap:
7     def __init__(self):
8         self._buckets = [None for i in range(10)]
9
10    def get_address(self, id_num):
11        return id_num % 10
12
13    def insert(self, employee):
14        address = self.get_address(employee.id_num)
15        self._buckets[address] = employee
16
17    def get_value(self, id_num):
18        address = self.get_address(id_num)
19        return self._buckets[address].name
20
21    def has(self, id_num):
22        address = self.get_address(id_num)
23        return self._buckets[address] is not None
```

O processo da hashmap

Quando duas chaves diferentes resultam no mesmo address

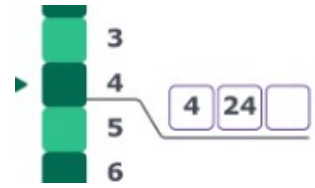
Fenômeno chamado de **colisão**.

Como lidar:

Separate chaining

Bucket vira lista de lista, ou seja cada posição guarda um array.

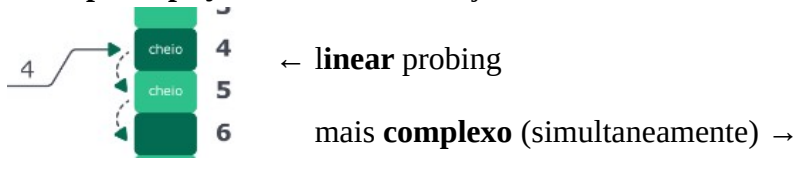
Inserir continua $O(1)$, mas outras opes perderam levemente performance.



Open Addressing com Linear Probing

O **endereço final não é conhecido**. Hashcode vira o ponto inicial de pesquisa em vez do final.

Busca por espaço vazio de diversos jeitos:



A classe Dict de Python

A classe Dict implementa a hashmap.

```
# Instanciando a classe Dict
employee_registry = {}

# Inserindo dados
# objeto[chave] = valor
employee_registry[14] = 'name1'
employee_registry[23] = 'name2'
employee_registry[10] = 'name3'
employee_registry[9] = 'name4'
print(employee_registry)

# Alterando o nome do id 10
# objeto[chave] = novo_valor
employee_registry[10] = 'name30'
print(f"Novo valor do id 10, após a atualização: {employee_registry[10]}")
```

//Apenas objetos **imutáveis** podem ser utilizados como **chave**. Ou seja, apenas aqueles objetos que depois de instanciados não podem ser alterados. Em Python, são:

- int;
- float;
- string;
- tuple;
- range;
- byte;
- frozenset.

→ Outros **métodos** para manipular dados.

→ Dict na [doc oficial](#).

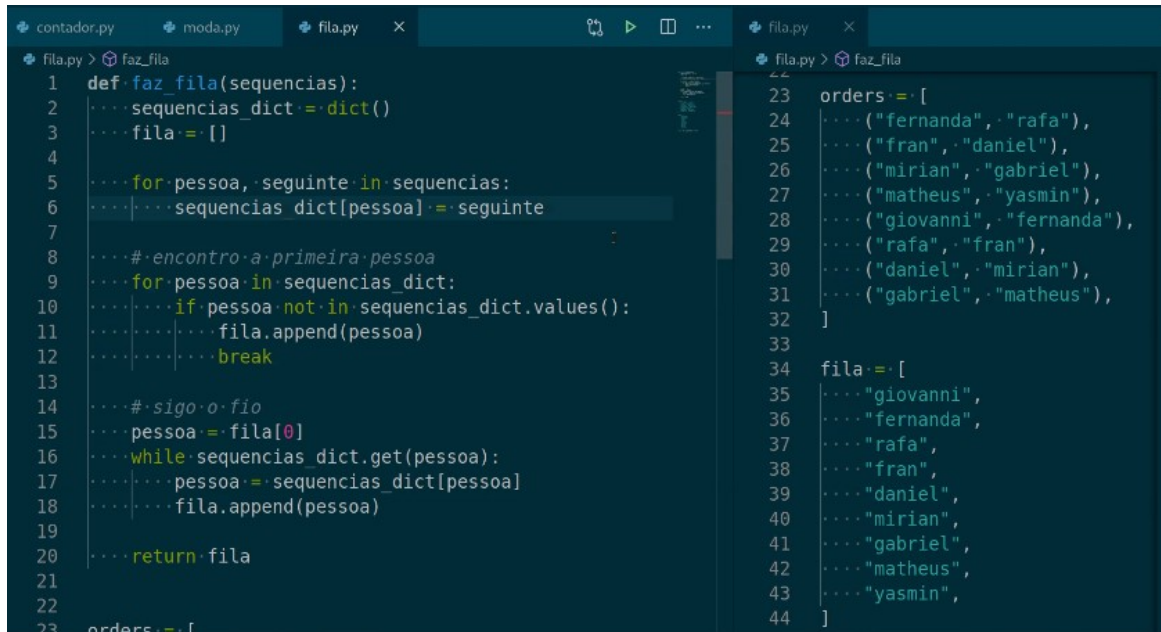
Resumo do conteúdo e resolução de problemas

Resolvendo:

- Encontrar o número mais frequente num array;
- Separar palavras de acordo com a sua letra inicial;
- Interseção entre listas.

Dicas diversas

- Diferença hashtable – hashmap;
- Dict comprehension, com exemplos.



```
1 def faz_filas(sequencias):
2     ...sequencias_dict=dict()
3     ...fila=[]
4
5     ...for pessoa, seguinte in sequencias:
6     ...sequencias_dict[pessoa]= seguinte
7
8     ...# encontro a primeira pessoa
9     ...for pessoa in sequencias_dict:
10    ...    if pessoa not in sequencias_dict.values():
11    ...        fila.append(pessoa)
12    ...        break
13
14    ...# sigo o fio
15    ...pessoa= fila[0]
16    ...while sequencias_dict.get(pessoa):
17    ...    pessoa= sequencias_dict[pessoa]
18    ...    fila.append(pessoa)
19
20    ...return fila
21
22
23 orders=[
24     ...("fernanda", "rafa"),
25     ...("fran", "daniel"),
26     ...("mirian", "gabriel"),
27     ...("matheus", "yasmin"),
28     ...("giovanni", "fernanda"),
29     ...("rafa", "fran"),
30     ...("daniel", "mirian"),
31     ...("gabriel", "matheus"),
32 ]
33
34 fila=[
35     ..."giovanni",
36     ..."fernanda",
37     ..."rafa",
38     ..."fran",
39     ..."daniel",
40     ..."mirian",
41     ..."gabriel",
42     ..."matheus",
43     ..."yasmin",
44 ]
```

- <https://replit.com/> para codar online.
- booleanos usam pouco espaço

2) Estrutura de dados II - Set

Python particularmente poderosa e nativa para conjuntos (sets)

Conceito de conjuntos

Muitas entidades do mundo real podem ser modeladas como conjuntos.

Coleção bem definida de elementos. Pode definir por listagem ou descrição dos elementos.

Listagem explícita:

A = {1, 2, 3, 4, 5, 6}

Descrição dos elementos

B = {x | x é um número inteiro tal que 0 < x ≤ 6}

Iguais se cada elemento de A pertence a B e se cada elemento de B pertence a A, sendo que nem a ordem nem a repetição importa.

Diferentes operações sobre conjuntos:

- **união**

set1.union(set2) para achar set1 | set2

- **intersecção**

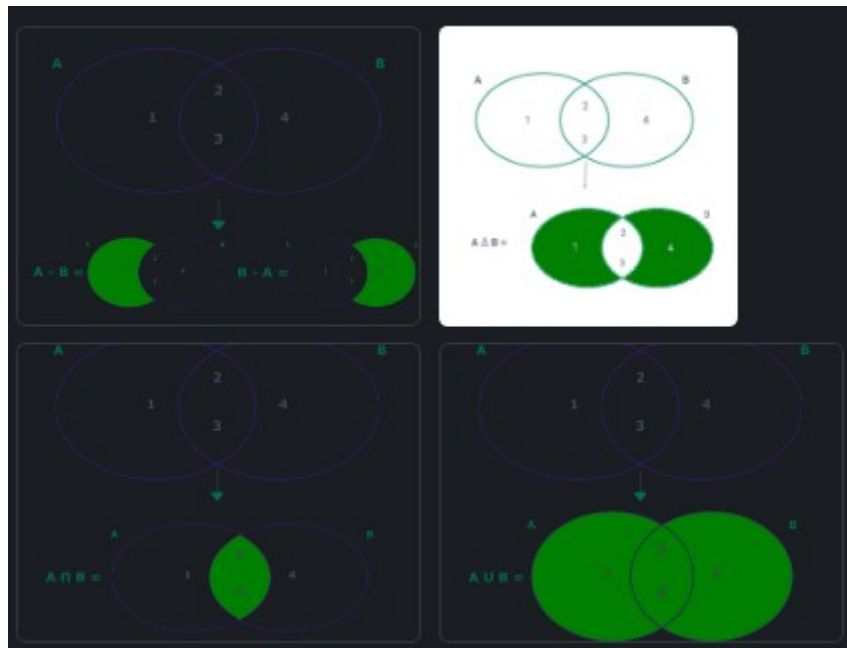
set1.intersection(set2) para achar set1 & set2

- **diferença**

- **diferença simétrica** (exclusivamente em uma lista ou outra)

- **subconjunto**

```
# Principais operações com set
print(set_a.union(set_b))
print(set_a.intersection(set_b))
print(set_a.difference(set_b))
print(set_b.difference(set_a))
```



// dif, dif sim, intersection, union

Formas de se representar conjuntos

Vetores

Caso relevante: **conjunto pequeno de numbers pequenos.**

Vetor de 0 a n iniciado como false, preencher para true quando tiver esse elemento no set.



Vetor de Booleanos

Hasmaps (dict)

Para casos onde vetores falham ou ficam pouco performantes.

```

exemplo.py X
home > cristiano > Desktop > exemplo.py > ...
1 A = {"Teste": True, 2: True, 3: True}
2
3 print("Teste" in A)
4

DEBUG CONSOLE  PROBLEMS  OUTPUT  TERMINAL
cristiano@trybe ~/Desktop$ cd /home/cristiano/
cristiano@trybe ~/Desktop$ python3 exemplo.py
True

```

A classe Set

Por baixo dos panos, a classe Set é uma **modificação da classe Dict**.

Set é uma **coleção não ordenada de objetos imutáveis únicos**.

Set **não guarda valores e é mutável**.

→ Frozen set

Para criar **sets de set**, o elemento de dentro precisa ser um frozenset (objeto idêntico a set, porém imutável).

→ Operações básicas

*# Podemos **instanciar um set** vazio ou inicializar com valores de um objeto iterável, como uma lista*
*conjuntoA = **set()***

*# Ao inicializar com valores de uma lista, os **valores duplicados serão desconsiderados** e a ordem de inserção será perdida.*

*conjuntoB = **set([1, 1, 2, 3, 3, 3])***

*# **Add** - adiciona o elemento ao conjunto*

*conjuntoA.**add**(5)*

*conjuntoA.**add**(3)*

*conjuntoA.**add**(0)*

*# **sets admitem objetos mistos**. Ou seja, admitem ter **_strings_** com **_ints_** dentro de um mesmo objeto, por exemplo.*

*conjuntoA.**add**('elemento')*

*# **Temos 2 jeitos de remover elementos**:*

*# - **remove()** causa erro caso o elemento não esteja no set;*

*# - **discard()** não causa erro caso o elemento não esteja no set.*

Não vai dar erro

*conjuntoB.**remove**(3)*

Vai dar erro pois já removemos esse elemento e set não guarda duplicatas

*conjuntoB.**remove**(3)*

Não vai dar erro

*conjuntoB.**discard**(3)*

*# **pop()** remove e retorna um elemento aleatório do set*

set é um objeto iterável, mas não conseguimos garantir em que ordem os elementos serão acessados.

*# A função **pop()** é útil quando queremos trabalhar com algum elemento do set, mas não sabemos de antemão quais elementos estão dentro dele.*

*some_element = conjuntoA.**pop**()*

*# **clear()** remove todos os itens do set*

*conjuntoB.**clear**()*

Consulta

A consulta é feita com o operador "in"

*if 2 **in** conjuntoA:*

print("2 está em A")

if 7 not in conjuntoA:

print("7 não está em A")

Resumão + resolução de problemas

Resolvendo:

- Números repetidos
- Dados de sorte (soma 7)

```
def get_repeated(nums):  
    seen_before = set()  
    repeated = set()  
  
    for num in nums:  
        if num in seen_before:  
            repeated.add(num)  
        else:  
            seen_before.add(num)  
  
    return repeated
```

```
def get_sevens(rolls):  
    seen_before = set()  
    answer = []  
  
    for roll in rolls:  
        if 7-roll in seen_before:  
            answer.append((7-roll, roll))  
            seen_before.discard(7-roll)  
        else:  
            seen_before.add(roll)  
  
    return answer
```

Dicas diversas

- Algoritmo rápido pode ser ruim? Sim - em caso de proteger info sensível, queremos propositalmente algo lento!
 - Sempre que pensar que pode usar um *dict* com valores *booleanos*, é um bom caso de *set*.
-

3) Projeto – Restaurant Orders

Dicas diversas do projeto:

- Extension VsCode [edit csv](#) ;
 - `from collections import defaultdict ;`
 - [os](#) & [errno](#)
 `raise FileNotFoundError(
 errno.ENOENT, os.strerror(errno.ENOENT), smthg_you_want
)`
-