

②

ハッシュテーブルの基本

ハッシュテーブルを使って、集合を実現する方法を学習します。

基本的なアイデア

- 辞書や集合をリストで実現すると、何故非効率的なのでしょうか？
- → 要素がリストのどこにあるかが明らかではなく、要素を探すために線形探索を行うためです



- そこでハッシュテーブルでは「要素がリストのどこにあるか」が、「すぐに分かる」仕組みを導入します

ハッシュテーブルの基本

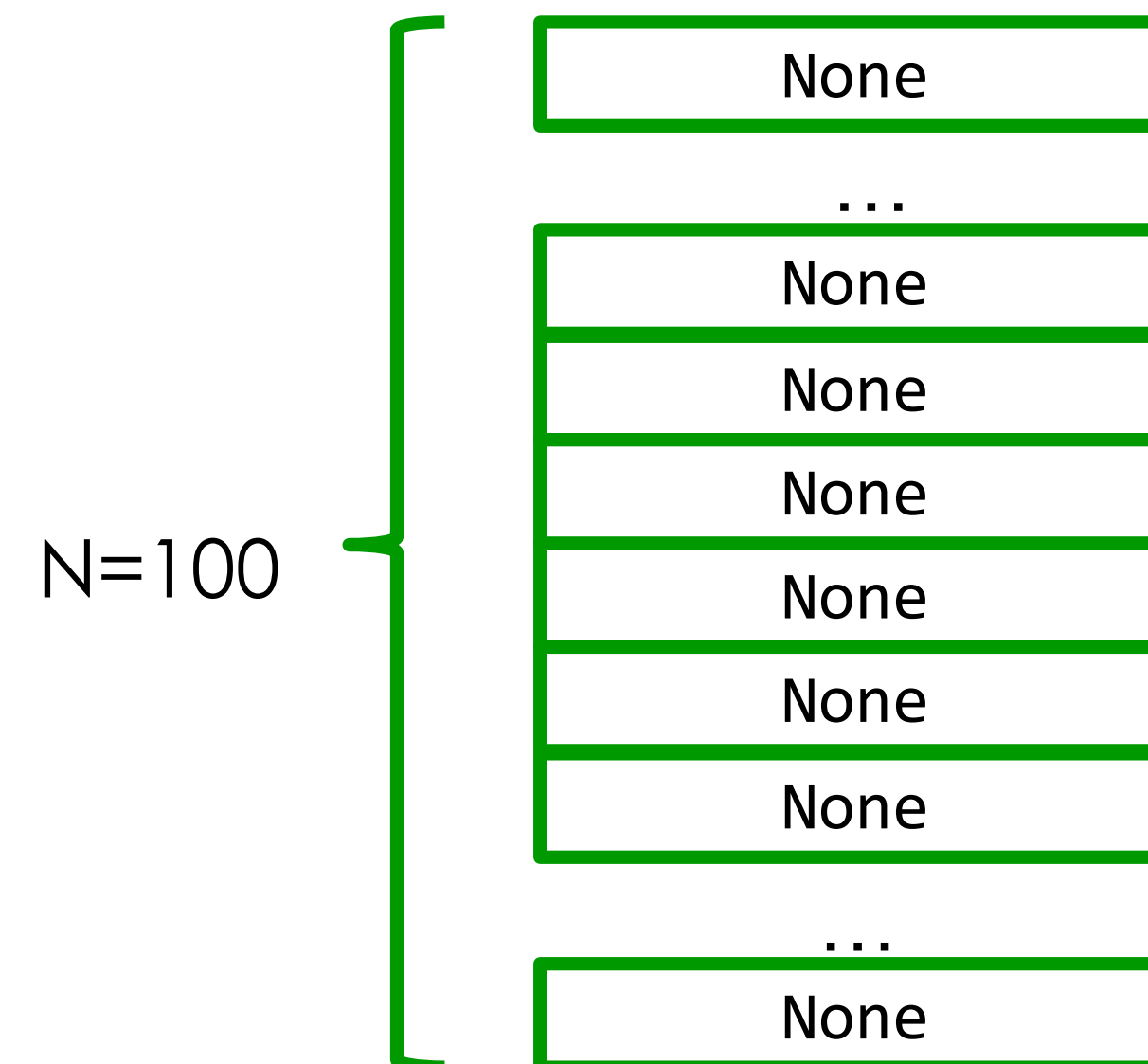
- 集合の要素が十分に入りきる、長さ N の配列を用意します
 - この配列を「ハッシュテーブル」と呼びます
 - 既に学習した通り、Pythonのリストは配列なので、ここは「リスト」と読み替えても大丈夫です
- 要素 x を引数にとり、 0 以上 N 未満の整数を返す関数 $h(x)$ を用意します
 - この関数を「ハッシュ関数」、返された整数を「ハッシュ値」と呼びます
- ハッシュテーブルに要素 x を追加する時には、配列の $h(x)$ 番目に追加します

簡単な例

- ここでは、整数の集合をハッシュテーブルに格納することを考えます
 - 配列の長さ **N** を 100 とします
 - ハッシュ関数は、整数を100で割った余りを返す関数とします

```
N = 100
table = [None] * N

def h(x):
    return x % N
```



要素を追加するには？

- 要素を追加する際には、追加する要素のハッシュ値を計算し、配列の該当箇所に格納します
 - 既に配列に値が入っている場合は例外が発生します

```
def add(table, x):  
    hash_value = h(x)  
    if table[hash_value] != None:  
        raise Exception('Hash collision!')  
    table[hash_value] = x
```

```
add(table, 175248)  
add(table, 721501)  
add(table, 989332)  
add(table, 851274)  
add(table, 978317)
```

```
table
```

N=100

0	None
1	721501
...	
17	978317
	None
...	
32	989332
...	
48	175248
	None
...	
74	851274
...	
99	None

要素を検索するには？

- 要素を検索する際には、検索する要素のハッシュ値を計算し、配列の該当箇所と比較します

```
def contains(table, x):  
    hash_value = h(x)  
    return table[hash_value] == x
```

```
contains(table, 175248)
```

True

48番目の要素と一致するため含まれている

```
contains(table, 100048)
```

False

48番目の要素と一致しないため含まれていない

```
contains(table, 101010)
```

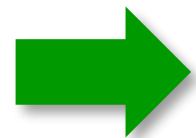
False

10番目の要素はNoneなので含まれていない

N=100

0	None
1	721501
...	
17	978317
	None
...	
32	989332
...	
48	175248
	None
...	
74	851274
...	
99	None

ハッシュテーブルと衝突

- 少し考えればわかりますが、たまたまハッシュ値が同じ値になることがあります
 - このことを「衝突」と呼びます
 - 今回のサンプルでは、例外を発生するようにしています
- 理想的には、なるべく衝突が起こらないようにすることが望めますが、実際には衝突することがあります
- そのため、ハッシュテーブルは衝突が起こることを前提に設計することが必要です（後で学習します）
- なお衝突がおきなければ、追加、削除、検索（所属の判定）とも、要素数によらず決まった時間で実現できます  $O(1)$

ハッシュ関数について

- ここでは簡単のため、整数の集合を考えましたが、実際には整数以外の値も扱える必要があります
- この場合は、整数以外についてもハッシュ値を計算できるようなハッシュ関数を用意する必要があります
- Pythonの場合には、あらかじめオブジェクトのハッシュ値を計算する関数 `hash` が用意されています

```
hash('apple')
```

```
hash('orange')
```

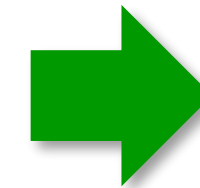
```
hash('banana')
```


ハッシュ関数について

- したがって、先ほど定義した関数 $h(x)$ を以下のようにすることで、文字列などについても集合に追加できるようになります

```
N = 100
table = [None] * N

def h(x):
    return hash(x) % N
```



```
add(table, 'apple')
add(table, 'banana')
```

```
contains(table, 'apple')
```

True

```
contains(table, 'orange')
```

False