④

衝突への対応

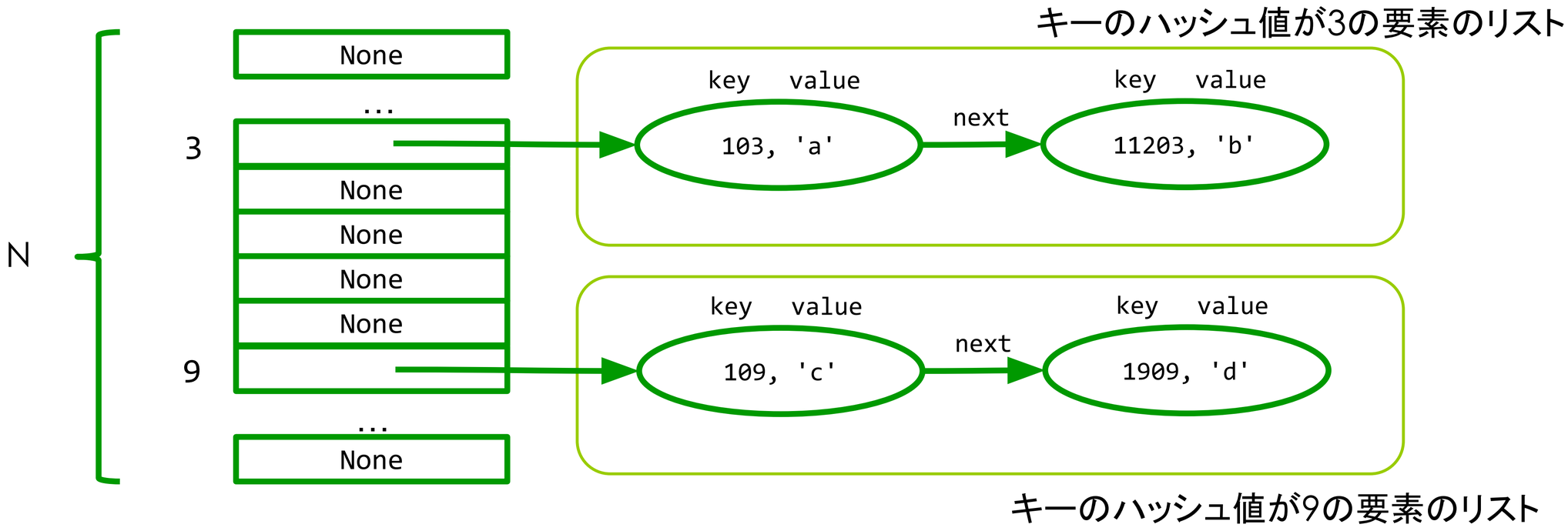
最後に、ハッシュテーブルにおける衝突への対応について学習します。

衝突が起きた場合の対応

- 先ほど学習したように、ハッシュテーブルでは衝突は起きうる事象のため、衝突が起きた場合の対応が必要です
- 一般的に、次の2種類の実装方法が用いられます
  - 連鎖法：衝突が起きても大丈夫なように、ハッシュテーブルにリンクリストを組み合わせる
  - オープンアドレス法：衝突が起きた場合には、ハッシュテーブルの他の空いている番地を利用する
- ここでは、先ほど実装した「辞書」を、連鎖法に対応させてみましょう

連鎖法の考え方

- 連鎖法では、ハッシュテーブルには、ハッシュ値ごとにリンクリストを格納します
  - 辞書の場合は、キーのハッシュ値が同一になる要素のリストとなります



Pythonで表現すると…

- 先ほど定義したクラス Node に、次要素へのリンクを表す属性 next を追加します

```
class Node:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None
```

```
dict_table = [None] * N
```

next追加します

次のセルも実行し、一旦テーブルを空にします

辞書に要素を追加するには？

- 引数で指定したキーが、既に登録されていた場合は...
  - この場合は、キーのハッシュ値に対応するリンクリストの中に既にノードがあるはずです

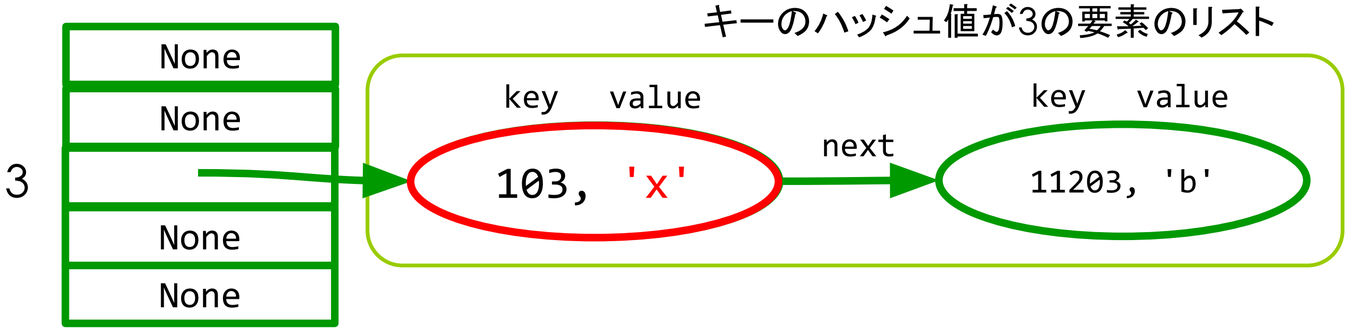
```
def put(table, key, value):
    hash_value = h(key)

    node = table[hash_value]
    while node != None:
        if node.key == key:
            node.value = value
            return
        node = node.next

    first_node = table[hash_value]
    new_node = Node(key, value)
    new_node.next = first_node
    table[hash_value] = new_node
```

ハッシュ値に対応するリンクリストを辿り、対応するキーがある場合には値を書き換えます

key=103, value='x' の場合...



辞書に要素を追加するには？

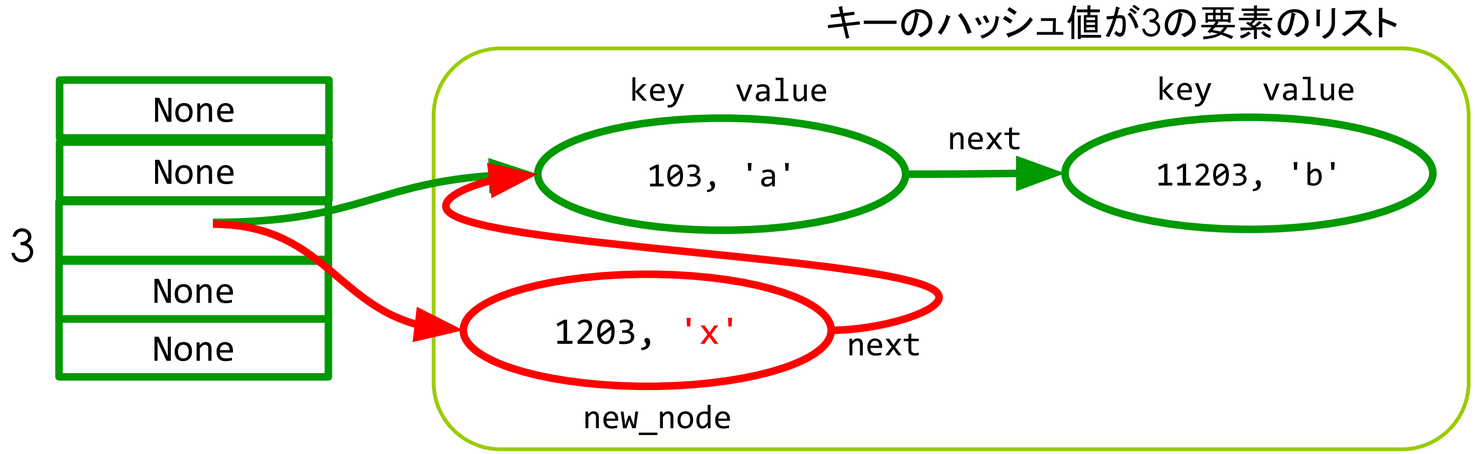
- 指定したキーが登録されていない場合は…
  - この場合は、キーのハッシュ値に対応するリンクリストの先頭にノードを追加します

```
def put(table, key, value):
    hash_value = h(key)

    node = table[hash_value]
    while node != None:
        if node.key == key:
            node.value = value
            return
        node = node.next

    first_node = table[hash_value]
    new_node = Node(key, value)
    new_node.next = first_node
    table[hash_value] = new_node
```

key=1203, value='x' の場合…



ハッシュ値に対応するリンクリストの先頭にノードを新たに追加します

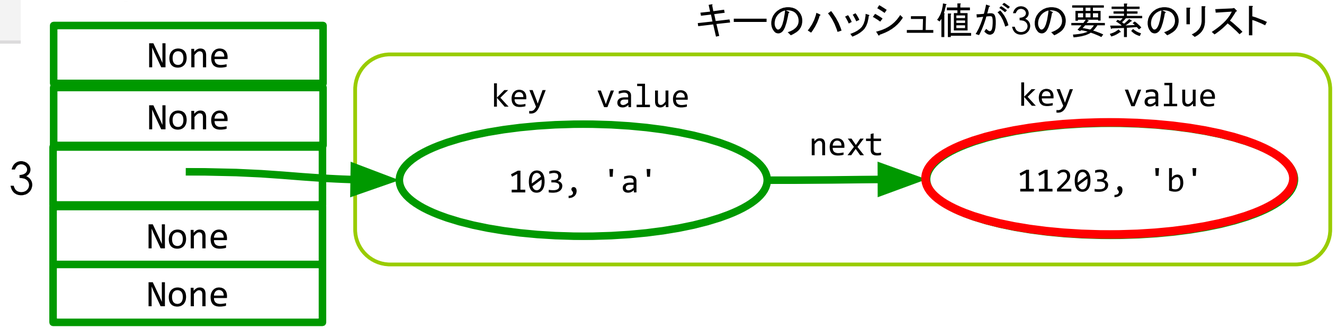
辞書の要素を検索するには？

- 辞書から要素を検索する際には、キーのハッシュ値に対応するリンクリストを辿り、キーが一致するノードを探します

```
def get(table, key):
    hash_value = h(key)
    node = table[hash_value]
    while node != None:
        if node.key == key:
            return node.value
        node = node.next
    return None
```

ハッシュ値に対応するリンクリストを辿り、対応するキーがある場合には値を返します

key=11203の場合...



return 'b'

このようにすると…

- このようにすると、衝突するキーがあっても、正しく値を追加したり検索することが、できるようになるはずです

```
put(dict_table, 103, 'a')
put(dict_table, 11203, 'b')
put(dict_table, 109, 'c')
put(dict_table, 1909, 'd')
```

```
get(dict_table, 103)
```

```
'a'
```

```
get(dict_table, 11203)
```

```
'b'
```

```
put(dict_table, 2003, 'e')
```

```
get(dict_table, 2003)
```

```
'e'
```

衝突を考慮したハッシュテーブルの計算量

- 一般に、要素数が十分に少なければ、追加、削除、所属の判定とも、計算量はO(1)と見なすことができます
- ただし、要素数が多いと衝突が起きやすくなるため、その分だけ計算量が大きくなります
- （参考）厳密には、以下のようになることが知られています（α = 要素数 / 配列サイズ）
  - 連鎖法の場合：　$O(1+\alpha)$
  - オープンアドレス法の場合（追加の場合）：　$O(\frac{1}{1-\alpha})$