

## 【K08】散列的课堂练习

解决思路：

当负载因子超过给定数值以后，扩大列表长度，同时 key 也会增大，之后，在进行 put 操作时，保证之后存入的位置一定在新开辟的部分，这样用新旧 key 存入的数就不会发生位置上的重叠。在进行提取时，一次用新、旧 key（从所给定的素数表中依次取用），查找即可。本伪代码忽略了一些和主题思路无关的细节。

这样做会浪费原先表中没有存储的空间，但只要负载因子设置大一些，结合实际问题，相比节省下的算力，可以认为是可以被忽略的。

```
primeNumlist = [11,13,17,19,23,29,31]#素数表，存储着自 11 之后的大量素数，本伪代码只是示意  
  
class HashTable:  
  
    def __init__(self):
```

```
ct = 0

global ct

self.size = primeNumlist[ct] #从素数表第一个开始取

self.slots = [None] * self.size

self.data = [None] * self.size


def hashfunction(self,key):

    return key % self.size


def rehash(self,oldhash):

    return (oldhash+1)%self.size


def isFull(self):#负载因子的判断 本次设定边界为 0.8

    return sum(x is not None for x in self.slots)/self.size > 0.8


def put(self,key,data):

    if not HashTable.isFull():#假如负载因子没有到达边界条件

        hashvalue = self.hashfunction(key)

        if self.slots[hashvalue] == None:

            self.slots[hashvalue] = key

            self.data[hashvalue] = data #如果相等 则替换

        else:
```

```

        nextslot = self.rehash(hashvalue)

        while self.slots[nextslot] != None and \

            self.slots[nextslot] != key:

            nextslot = self.rehash(nextslot)

        if self.slots[nextslot] == None:

            self.slots[nextslot] = key

            self.data[nextslot] = data

        else:

            self.data[nextslot] = data #替换

    else:

        ct += 1

        HashTable.self.size = primeNumlist[ct] #列表规模扩大

        hashvalue = self.hashfunction(key)

        while hashvalue <= HashTable.self.primeNumlist[ct-1]:#假如列表扩增以后,

            # 算出的位置仍然在前半部分的边界内, 则不管此时有没有被填满,

            # 直接向后探测, 直到比原来列表的长度长, 这样可以方便之后的提取

            #这样的存储方式, 可以保证在原先的和之后的 key 来计算提取时, 前、后的列表

数据是割裂开的,

            # 假如用一个 key 查找失败, 则可以用其余的 key 查找, 且不会发生重复

            hashvalue += 1

```

```
if self.slots[hashvalue] == None:

    self.slots[hashvalue] = key

    self.data[hashvalue] = data #如果相等 则替换

else:

    nextslot = self.rehash(hashvalue)

    while self.slots[nextslot] != None and \

        self.slots[nextslot] != key:

        nextslot = self.rehash(nextslot)

    if self.slots[nextslot] == None:

        self.slots[nextslot] = key

        self.data[nextslot] = data

    else:

        self.data[nextslot] = data #替换
```