Wongnophadol_Atit_Assignment6

December 26, 2018

0.1 Atit Wongnophadol

1 A Simple Hash Table with Linear Probing

In this exercise, your task is to implement a variant of a hash table. Several simplifications will make this task easier. First, your table will have a fixed size; it will not be capable of expanding to fit more data. Your table will only accept strings as keys, though values may be any Python object. Finally, you will use linear probing to resolve collisions.

Create a class, MyTable, with the following properties:

Table: Your table will have a fixed size, which you should pass in as a parameter to the initializer. Specifically, you should create a list to store keys (named keys) and a list to store values (named values). All items in these lists should initially be set to the object None.

Keys and Values: The keys to your table will be strings. Values may be any python object.

Hashcode: Your class should convert each character in a key to its unicode code point (use python's ord function) and then simply sum them together.

Compression function: To ensure the results of your hashcode falls in the right range, use the mod operator (%) with the size of the hash table.

Collision resolution: You will use linear probing to resolve collisions. If a particular location in the table is filled, you move forward one space until an empty location is found. If you reach the end of the table, you cycle back to index 0.

Deletions: As with any open addressing system, deletions must be executed with care. Finding one item A may rely on the fact that item B was in a particular location when A was inserted. To get around this problem, you should store three types of objects in your list of keys. The object None indicates that a space has never been used. The special string "deleted" indicates that the space was used before but is now available. All other strings represent keys that have been stored in the table.

Inside your MyTable class, you must implement the following methods:

- __setitem__(key, value) - insert the given key-value pair into the table. If the given key is already in the table, replace the old value with the new value.
- __getitem__(key) - get the value that corresponds to the given key in the table.
- __delitem__(key) - remove the given key and its corresponding value from the table. Replace both with the special string "deleted".

Note that these are magic methods that should not be accessed directly, but will be called when indexing instances of your class with square brackets

In case __getitem__ is called with a key that is not in the table, return the string. "Key not in table."

Additionally, you should only access your keys list one index at a time and avoid looping through all items in the list whenever possible. This also means that you should not use operators like *in* that implicitely loop through all items in your list.

The following code demonstrates the proper use of the MyTable class. Make sure that your class replicates this behavior exactly.

```
In [1]: class MyTable():

            def __init__(self, size):
                self._n = size
                self.keys = size*[None]
                self.values = size*[None]

            def __setitem__(self, k, v):
                self._setvalue(k, v)

            def __getitem__(self, k):
                return self._getvalue(k)

            def __delitem__(self, k):
                self._delvalue(k)

            def _hash_function(self, k):
                i = 0
                hashed_k = 0
                while i < len(k):
                    hashed_k+=ord(k[i])
                    i+=1
                return hashed_k

            def _compression_function(self, k):
                return k%self._n

            def _collision_solution(self, k):
                return (k+1)%self._n

            def _setvalue(self, k, v):

                position = self._compression_function(self._hash_function(k))

                if self.keys[position] == None:
                    self.keys[position] = k
                    self.values[position] = v
                else:
                    if self.keys[position] == k:
                        self.values[position] = v  #replace
```

```
            else:
                position = self._collision_solution(position)
                while self.keys[position] != None and self.keys[position] != k:
                    position = self._collision_solution(position)
                if self.keys[position] == None:
                    self.keys[position] = k
                    self.values[position] = v
                else:
                    self.values[position] = v

    def _getvalue(self, k):

        initial_position = self._compression_function(self._hash_function(k))

        value = None
        stop = False
        found = False
        position = initial_position
        while self.keys[position] != None and not found and not stop:
            if self.keys[position] == k:
                found = True
                value = self.values[position]
            else:
                position=self._collision_solution(position)
                if position == initial_position:
                    stop = True

        if value is None:
            return "Key not in table"
        else:
            return value

    def _delvalue(self, k):

        initial_position = self._compression_function(self._hash_function(k))

        stop = False
        found = False
        position = initial_position
        while self.keys[position] != None and not found and not stop:
            if self.keys[position] == k:
                found = True
                self.keys[position] = "deleted"
                self.values[position] = "deleted"
            else:
                position=self._collision_solution(position)
                if position == initial_position:
                    stop = True
```

```
In [2]: m = MyTable(5)
        # The following keys all hash to the same index.
        m["a"] = "apple"
        m["f"] = "butter"
        m["k"] = "yummy"
        print("Current keys in m:", m.keys)
```

```
Current keys in m: [None, None, 'a', 'f', 'k']
```

```
In [3]: # "p" also hashes to the same place.
        # Your class should detect that it's not in the table
        # without scanning through the entire keys list.
        print("m['p']:", m["p"])
```

```
m['p']: Key not in table
```

```
In [4]: # We can access key "k"
        print("m['k']:", m["k"])
        # Even if we remove "f"
        del m["f"]
        print("m['k']:", m["k"])
        print("Current keys in m:", m.keys)
```

```
m['k']: yummy
m['k']: yummy
Current keys in m: [None, None, 'a', 'deleted', 'k']
```

```
In [7]: # Even after removing "f", we can overwrite "k"
        m["k"] = "newstuff"
        print("Current keys in m:", m.keys)
        print("Current values in m:", m.values)
```

```
Current keys in m: [None, None, 'a', 'deleted', 'k']
Current values in m: [None, None, 'apple', 'deleted', 'newstuff']
```