Daily Coding Problem #97

Problem

This problem was asked by Stripe.

Write a map implementation with a get function that lets you retrieve the value of a key at a particular time.

It should contain the following methods:

- `set(key, value, time)`: sets key to value for t = time.
- `get(key, time)`: gets the key at t = time.

The map should work like this. If we set a key at a particular time, it will maintain that value forever or until it gets set at a later time. In other words, when we get a key at a time, it should return the value that was set for that key set at the most recent time.

Consider the following examples:

```
d.set(1, 1, 0) # set key 1 to value 1 at time 0
d.set(1, 2, 2) # set key 1 to value 2 at time 2
d.get(1, 1) # get key 1 at time 1 should be 1
d.get(1, 3) # get key 1 at time 3 should be 2

d.set(1, 1, 5) # set key 1 to value 1 at time 5
d.get(1, 0) # get key 1 at time 0 should be null
d.get(1, 10) # get key 1 at time 10 should be 1

d.set(1, 1, 0) # set key 1 to value 1 at time 0
d.set(1, 2, 0) # set key 1 to value 2 at time 0
d.get(1, 0) # get key 1 at time 0 should be 2
```

Solution

One possible way to solve this question is using a map of maps, where each key has its own map of time-value pairs. That would mean something like:

```
 {
     key: {
         time: value,
         time: value,
         ...
     },
     key: {
         time: value,
         time: value,
```

```
        ...
    },
    ...
}
```

Also, if a particular time does not exist on the time-value map, we must be able to get the value of the nearest previous time (or null if doesn't have one). A sorted map would fit the bill, but python standard library doesn't have one. So, let's see how this map would look:

```
class TimeMap:
    def __init__(self):
        self.map = dict()
        self.sorted_keys_cache = None

    def get(self, key):
        value = self.map.get(key)
        if value is not None:
            return value
        if self.sorted_keys_cache is None:
            self.sorted_keys_cache = sorted(self.map.keys())
        i = bisect.bisect_left(self.sorted_keys_cache, key)
        if i == 0:
            return None
        else:
            return self.map.get(self.sorted_keys_cache[i - 1])

    def set(self, key, value):
        self.sorted_keys_cache = None
        self.map[key] = value
```

This is a map with a list of sorted keys. To find out the nearest previous time we use the binary search algorithm provided by the bisect.

Any write operation on this map wipes the key's cache, causing a full sort of the keys on the next `get` call, which in python's TimSort averages as O(n log n) complexity.

For mixed workloads, a more suitable approach is to use arrays under the hood. Something like this:

```
class TimeMap:
    def __init__(self):
        self.keys = []
        self.values = []

    def get(self, key):
        if self.keys is None:
            return None
        i = bisect.bisect_left(self.keys, key)
        if len(self.keys) == i:
            return self.values[i - 1]
        elif self.keys[i] == key:
            return self.values[i]
        elif i == 0:
            return None
        else:
            return self.values[i - 1]

    def set(self, key, value):
        i = bisect.bisect_left(self.keys, key)
        if len(self.keys) == i:
            self.keys.append(key)
            self.values.append(value)
```

```
        elif self.keys[i] == key:
            self.values[i] = value
        else:
            self.keys.insert(i + 1, key)
            self.values.insert(i + 1, value)
```

In this way, both `get` and `set` behave more predictable from the performance standpoint, it's just a binary search, and for `set` two array reallocations in the worst case.

The last missing part to solve this question is the first level map, which the code would look this:

```
class MultiTimeMap:
    def __init__(self):
        self.map = defaultdict(TimeMap)

    def set(self, key, value, time):
        self.map[key].set(time, value)

    def get(self, key, time):
        time_map = self.map.get(key)
        if time_map is None:
            return None
        else:
            return time_map.get(time)
```

Now each key can have its own TimeMap, initialized by defaultdict when needed.