Vishal Chugh

Redis

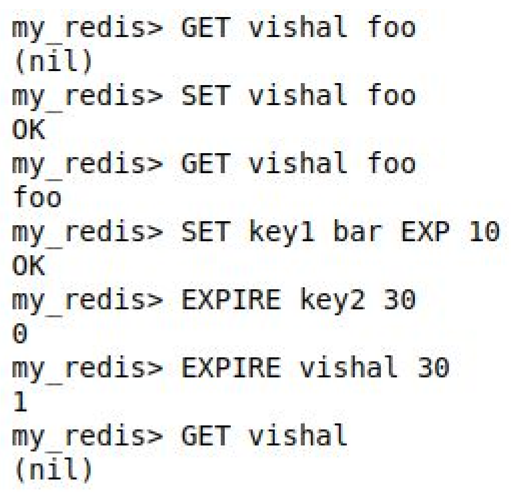
Introduction

Major points in the implementation of the given assignment are:

1. Since values can be either of type string or a sorted set, two classes are created one that for string and one that for sorted set each containing various variables and functions used for the particular data type.
2. In the String class threre are mainly two functions: set() and get().
3. In the Sorted Set class various variables such as scores, values, size of the set of a particular key etc are used and the functions implemented are particular to that of sorted sets such as zadd, zrank and zrange alongwith finding the index using Binary Search.
4. A Hash Table class is implemented that acts as a helper class to String and Sorted Set class and calls various functions of these classes depending on the command given. It has a dictionary which is the basic data structure to store the (key, value) pairs.
5. A dictionary to store the **expiry time of a key** is also created and a thread runs to check every 10 ms whether any of the keys has expired and pops that key from the dictionary.
6. A server is created which receives a command from the client and processes it and returns the values to the client.
7. The various commands implemented are:
    - **SET:** It takes as input a key and a value to be assigned to that key. An object of the custom String class is created for the value and the hash_table dictionary is updated for the given key. It returns "OK" if key is successfully inserted.

EXP can be added as an option followed by a numerical value to indicate expiry time of that key.

- **GET:** It takes as input a key and returns the value assigned to that key. If the key is not present it returns “(nil)”
- **EXPIRE:** It allows to set a timeout for a key already present. It returns 0 if key is not present and 1 if key is present.



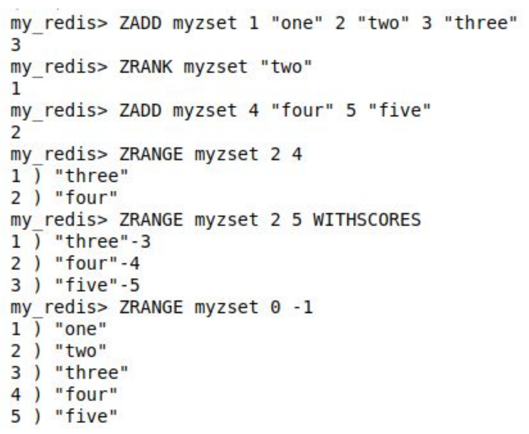
```
my_redis> GET vishal foo
(nil)
my_redis> SET vishal foo
OK
my_redis> GET vishal foo
foo
my_redis> SET key1 bar EXP 10
OK
my_redis> EXPIRE key2 30
0
my_redis> EXPIRE vishal 30
1
my_redis> GET vishal
(nil)
```

- **ZADD:** It allows to add a sorted set to a key and the input given is the key along with the list of (value, scores) to be added. It returns the number of new scores added.

  We used binary search to find the index where the score needs to be added as scores need to be sorted.

- **ZRANK:** It gives the rank of the given key w.r.t zero indexing and here also binary search is used to find the index of the given key.
- **ZRANGE:** It gives the values in the specified range in the Sorted Set. Since the list of (value, scores) is already sorted it becomes quite simple to get a particular range.

  WITHSCORES can be used as an option to display the scores as well.



```
my_redis> ZADD myzset 1 "one" 2 "two" 3 "three"
3
my_redis> ZRANK myzset "two"
1
my_redis> ZADD myzset 4 "four" 5 "five"
2
my_redis> ZRANGE myzset 2 4
1 ) "three"
2 ) "four"
my_redis> ZRANGE myzset 2 5 WITHSCORES
1 ) "three"-3
2 ) "four"-4
3 ) "five"-5
my_redis> ZRANGE myzset 0 -1
1 ) "one"
2 ) "two"
3 ) "three"
4 ) "four"
5 ) "five"
```

Questions

1. Python was the language chosen by me as it is very dynamic in terms of data types to be used and since I wanted to focus on the architecture of the system rather than focussing on various data types to be used I preferred Python.

2. Further improvements that can be made are:

    a. Could have implemented multithreading to allow multiple users access the data concurrently. Currently only one user is allowed to give command to the system.

    b. For small length keys since they can only be strings a **Trie data structure** could have been used for faster access of data with the scores stored in each node of the trie since it provides for faster access of strings with common prefixes. But since it was specified in the documentation of Redis that it supports a string of length 512 MB, I gave up on trying this approach.

    c. More commands could have been implemented but given the duration and a lot of coursework to complete I could not implement more of the Redis commands.

3. The data structures used are:

   a. **Dictionary**: For mapping each key to its value which can be a string or a sorted set. A dictionary allows faster access and retrieval of data as compared to other data structures.
   b. **Lists**: To store the list of (key, value) pairs for a sorted set.
   c. **Tuple**: To store the (value, score) of a sorted set.

4. No, my implementation does not support multithreading as I read in the documentation of Redis:

> *"Redis is, mostly, a single-threaded server from the POV of commands execution (actually modern versions of Redis use threads for different things). It is not designed to benefit from multiple CPU cores. People are supposed to launch several Redis instances to scale out on several cores if needed. It is not really fair to compare one single Redis instance to a multi-threaded data store."*

Although I thought of implementing multithreading to allow multiple users to access and work on data concurrently but such scenarios can lead to Race Condition and will need some Mutual Exclusion method to take care of such a situation.