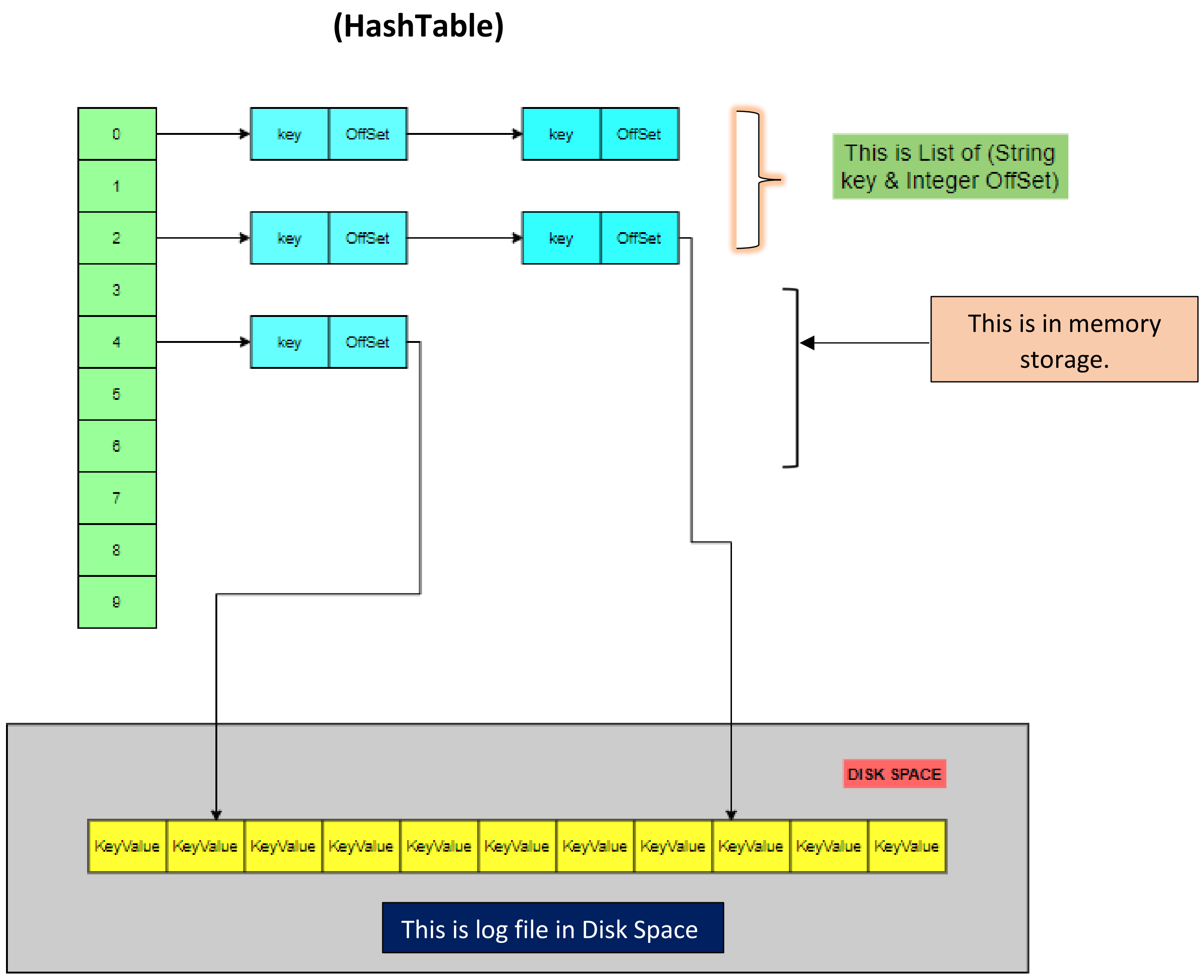
Report Lab Project 1-B

[Shashank Shekhar] [2/24/2021] [Dr. Song Jiang]


(HashTable)
0
1
2
3
4
5
6
7
8
9
key
OffSet
This is List of (String key & Integer OffSet)
This is in memory storage.
DISK SPACE
KeyValue
This is log file in Disk Space

The HashTable in the code is used to store keys and their offset values.

```java
public static Hashtable<Integer, List<Entry>> KeyOffSetPair
= new Hashtable<>();
```

Values are of type List<Entry> which has String Key and Integer offSet.

```java
public class Entry {
    private String key;
    private Integer offSet;
}
```

Put()

When the *"put()"* method is called of the class *PutKeyValue* it adds *"key"* & *"Value"* in the logfile in its tail.

```
public static Integer put(String logfile, String key, String value)
```

Note: If there is a duplicate key which is already present in our log file, we won't update its value.

We will add the current key with its value in the logfile in the tail & update the HashTable with new entry.

Get()

(1) The HashValue is computed to get the desired *List<Entry>*.

```
int HashValue = getValueHash(Integer.parseInt(key.substring(key.length()-1)));
```

```
public static int getValueHash(Integer key) {
    Integer hashVal = null;

    if (key != null) {
        hashVal = key % 10;
    }
    return hashVal;
}
```

(2) When the *"get()"* is method is called of the class *GetKeyValue* it first checks our hashtable *keyOffSetPair* contains the computed hashvalue otherwise it returns 0.

```
public static Integer getOffSet(String key) throws IOException {

    int hashValue = getValueHash(Integer.parseInt(key.substring(key.length()
- 1)));

    if (KeyOffSetPair.containsKey(hashValue)) {

        List<Entry> keyOffSet_List = KeyOffSetPair.get(hashValue);

        if (keyOffSet_List != null) {
            return getOffSet(keyOffSet_List, key);
        }
    }

    return 0;

}
```

```
public static Integer getOffSet(List<Entry> entries, String key) {

    Integer maxOffset = 0;

    for (Entry entry : entries) {
        if (entry.getKey().equals(key) && maxOffset < entry.getOffSet()) {
            maxOffset = entry.getOffSet();
        }
    }

    return maxOffset;
}
```

(3) We iterate over the list of Entries to check whether our current key matches with the keys present in the list, if a match is found, we get the OffSet value which gives us the location of our key in the logfile (which is line number in our logfile).

Design Challenges

(1) If we have to update KV item in-place in the log for a "put", then we will have to iterate over the whole logfile to find that particular key and update its value. The whole operation time will not be efficient in terms of time.

(2) Whenever we have to delete the key:
   a) We get the desired **hash-value** based on the key.
   b) If the hash table **does not contain** the hash-value or **entries** for that hash-value that means table doesn't exist in logfile and will return without doing any operation.
   c) If the hash table **contains** the hash-value and corresponding entries, then we get the list of entries and iterate over the list to get the corresponding **offset.** This **offset** denotes the **line number in log file** where the key value combination is present. We simply remove the content from that line in the logfile and delete the corresponding entry from the hash-table as well. **Note:** If there are multiple entries (for the same key) we have to the latest entry from the logfile. So, when we are iterating over the list of entries, we will get the maximum offset that will be deleted.

(3) If we don't save the HashTable, we can recover the index from out logfile which contains the key value pairs. We can reconstruct our HashTable from the logfile. We will iterate over the log file and from the key we will get our HashValue, which will be our key for the HashTable *keyOffSetPair* and the *OffSet* value will be the line number at which key value is present.

(4) If we save the index to the disk as an index file, below will be the format of the file, where {5, 4, 3, 1} are the keys of HashTable & Values are {Key, OffSet}.

```
5 => [{key0000000046265 : 7}],
4 => [{key0000000058594 : 4},{key0000000068534 : 6}],
3 => [{key0000000095583 : 5}],
1 => [key0000000090201 : 2}, {key0000000015611 : 3}]
```