3/26/25, 3:45 PM consistentHash

```
1
     package main
 2
 3
     import (
             "hash/crc32"
 4
 5
             "sort"
 6
 7
 8
    type Hash func(data []byte) uint32
 9
10
     type Map struct {
             hash
                       Hash
11
             replicas int
12
                       []int // Sorted
             keys
13
             hashMap map[int]string
14
     }
15
16
17
     func NewConsistentHash(replicas int, fn Hash) *Map {
             m := &Map{
18
19
                      replicas: replicas,
20
                      hash:
                      hashMap: make(map[int]string),
21
22
             if m.hash = nil {
23
24
                      m.hash = crc32.ChecksumIEEE
25
26
             return m
     }
27
28
29
     // IsEmpty returns true if there are no items available.
     func (m *Map) IsEmpty() bool {
30
             return len(m.keys) = 0
31
     }
32
33
     // Add adds some keys to the hash.
34
     //func (m *Map) Add(keys ... string) {
35
             for , key := range keys {
36
     //
     //
                      for i := 0; i < m.replicas; i \leftrightarrow \{
37
                              hash := int(m.hash([]byte(strconv.Itoa(i) + key)))
     //
38
39
     //
                              m.keys = append(m.keys, hash)
                              m.hashMap[hash] = key
40
     //
     //
                      }
41
42
     //
43
     //
             sort.Ints(m.keys)
44
     //}
45
     func (m *Map) Add(keys ... string) {
46
             for _, key := range keys {
47
                      hash := int(m.hash([]byte(key)))
48
                      m.keys = append(m.keys, hash)
49
                      m.hashMap[hash] = key
50
             }
51
             sort.Ints(m.keys)
52
53
     }
```

3/26/25, 3:45 PM consistentHash

```
54
55
    // Get gets the closest item in the hash to the provided key.
    func (m *Map) Get(key string) string {
56
             if m.IsEmpty() {
57
                     return ""
58
59
             }
60
             hash := int(m.hash([]byte(key)))
61
62
             // Binary search for appropriate replica.
63
             idx := sort.Search(len(m.keys), func(i int) bool { return m.keys[i] >
64
    hash })
65
             // Means we have cycled back to the first replica.
66
             if idx = len(m.keys) {
67
                     idx = 0
68
69
             }
70
             return m.hashMap[m.keys[idx]]
71
    }
72
73
```