```go
package main

import (
	"hash/crc32"
	"sort"
)

type Hash func(data []byte) uint32

type Map struct {
	hash     Hash
	replicas int
	keys     []int // Sorted
	hashMap  map[int]string
}

func NewConsistentHash(replicas int, fn Hash) *Map {
	m := &Map{
		replicas: replicas,
		hash:     fn,
		hashMap:  make(map[int]string),
	}
	if m.hash == nil {
		m.hash = crc32.ChecksumIEEE
	}
	return m
}

// IsEmpty returns true if there are no items available.
func (m *Map) IsEmpty() bool {
	return len(m.keys) == 0
}

// Add adds some keys to the hash.
//func (m *Map) Add(keys ...string) {
//	for _, key := range keys {
//		for i := 0; i < m.replicas; i++ {
//			hash := int(m.hash([]byte(strconv.Itoa(i) + key)))
//			m.keys = append(m.keys, hash)
//			m.hashMap[hash] = key
//		}
//	}
//	sort.Ints(m.keys)
//}

func (m *Map) Add(keys ...string) {
	for _, key := range keys {
		hash := int(m.hash([]byte(key)))
		m.keys = append(m.keys, hash)
		m.hashMap[hash] = key
	}
	sort.Ints(m.keys)
}
```

```go

// Get gets the closest item in the hash to the provided key.
func (m *Map) Get(key string) string {
	if m.IsEmpty() {
		return ""
	}

	hash := int(m.hash([]byte(key)))

	// Binary search for appropriate replica.
	idx := sort.Search(len(m.keys), func(i int) bool { return m.keys[i] > hash })

	// Means we have cycled back to the first replica.
	if idx == len(m.keys) {
		idx = 0
	}

	return m.hashMap[m.keys[idx]]
}

```