# [全 O(1) 的数据结构](https://leetcode-cn.com/explore/interview/card/bytedance/245/data-structure/1033/)

## 题目

实现一个数据结构支持以下操作：

* Inc(key) - 插入一个新的值为 1 的 key。或者使一个存在的 key 增加一，保证 key 不为空字符串。
* Dec(key) - 如果这个 key 的值是 1，那么把他从数据结构中移除掉。否者使一个存在的 key 值减一。如果这个 key 不存在，这个函数不做任何事情。key 保证不为空字符串。
* GetMaxKey() - 返回 key 中值最大的任意一个。如果没有元素存在，返回一个空字符串""。
* GetMinKey() - 返回 key 中值最小的任意一个。如果没有元素存在，返回一个空字符串""。

挑战：以 O(1) 的时间复杂度实现所有操作。

## 解题思路

1. 设计一个 Bucket 保存所有值为 value 的 key
2. 并且有临近 value 的 Bucket 指针

class AllOne {

 /\*\* Initialize your data structure here. \*/
 public AllOne() {

 }

 private static class Bucket {
 private int value;
 private Set<String> keys = new HashSet<>();
 private Bucket next;
 private Bucket pre;

 public Bucket(int value) {
 this.value = value;
 }

 @Override
 public String toString() {
 return "Bucket{" +
 "value=" + value +
 ", keys=" + keys +
 '}';
 }
 }

 private Map<String, Bucket> data = new HashMap<>();
 private List<Bucket> bucketList = new ArrayList<>();

 /\*\*
 \* Inserts a new key <Key> with value 1. Or increments an existing key by 1.
 \*/
 public void inc(String key) {
 if (data.containsKey(key)) {
 Bucket bucket = data.get(key);
 bucket.keys.remove(key);

 if (bucket.next == null) {
 bucket.next = new Bucket(bucket.value + 1);
 bucket.next.pre = bucket;
 bucketList.add(bucket.next);
 }

 bucket.next.keys.add(key);
 data.put(key, bucket.next);
 } else {
 if (bucketList.size() == 0) {
 bucketList.add(new Bucket(1));
 }

 Bucket bucket = bucketList.get(0);
 bucket.keys.add(key);
 data.put(key, bucket);
 }
 }

 /\*\*
 \* Decrements an existing key by 1. If Key's value is 1, remove it from the data structure.
 \*/
 public void dec(String key) {
 if (!data.containsKey(key)) {
 return;
 }

 Bucket bucket = data.get(key);
 if (bucket.pre == null) {
 bucket.keys.remove(key);
 data.remove(key);
 } else {
 bucket.keys.remove(key);
 bucket.pre.keys.add(key);
 data.put(key, bucket.pre);
 }
 }

 /\*\*
 \* Returns one of the keys with maximal value.
 \*/
 public String getMaxKey() {
 if (bucketList.size() == 0) {
 return "";
 }

 for (int i = bucketList.size() - 1; i >= 0; i--) {
 Bucket bucket = bucketList.get(i);

 if (!bucket.keys.isEmpty()) {
 Iterator<String> iterator = bucket.keys.iterator();
 if (iterator.hasNext()) {
 return iterator.next();
 } else {
 return "";
 }
 }
 }

 return "";
 }

 /\*\*
 \* Returns one of the keys with Minimal value.
 \*/
 public String getMinKey() {
 if (bucketList.size() == 0) {
 return "";
 }

 for (Bucket bucket : bucketList) {
 if (!bucket.keys.isEmpty()) {
 Iterator<String> iterator = bucket.keys.iterator();
 if (iterator.hasNext()) {
 return iterator.next();
 } else {
 return "";
 }
 }
 }

 return "";
 }
}