算法作业第二章

1 Question 2-3

1.1 问题描述

设 `a[0:n-1]` 是已排好序的数组。请改写二分搜索算法,使得当搜索元素x不在数组中时,返回小于x的最大元素位置i和大于x的最小元素位置j。当搜索元素在数组中时，i和j相同,均为x在数组中的位置。

1.2 实现算法

首先，确定要查找的列表的左边界 `left` 和右边界 `right` 。然后，在一个循环中，不断将待查找区间缩小一半，直到找到目标值或者确定目标值不存在为止。如果目标值存在，返回目标值的前后索引位置；如果不存在，则返回目标值应该插入的位置。

```
函数 binsearch(list, key, length):
    left = 0
    right = length - 1

    while left <= right:
        mid = (left + right) // 2
        如果 list[mid] == key:
            返回 mid, mid
        否则如果 list[mid] > key:
            right = mid - 1
        否则如果 list[mid] < key:
            left = mid + 1

    如果 right < 0:
        返回 -100000, left
    否则如果 left > length - 1:
        返回 right, 100000

    返回 right, left
```

2 Question 2-9

2.1 问题描述

设 `T[0:n-1]` 是n个元素的数组。对任一元素x，设$S(x) = \{i|T[i] = x\}$。当$|S(x)| > n/2$时，称x为T的主元素。设计一个线性时间算法，确定$T[0 : n - 1]$是否有一个主元素。

2.2 实现算法

首先使用快速选择算法找到列表中的第 k 小的元素（如果列表长度为奇数，则找到中间位置的元素；如果列表长度为偶数，则找到中间偏右的元素）。然后统计该元素在列表中出现的次数，如果超过列表长度的一半，则返回 True，否则返回 False。

这段代码的算法思想是基于快速选择算法，其目的是查找列表中是否存在超过一半次数的元素。快速选择算法类似于快速排序，但是它只关注找到第 k 小或第 k 大的元素，而不对整个列表进行完全排序。

```
函数 partition(arr, left, right):
    pivot = arr[left]
    low = left + 1
    high = right

    当循环条件为真时:
        当 low <= high 且 arr[low] <= pivot 时:
            low += 1
        当 low <= high 且 arr[high] >= pivot 时:
            high -= 1
        如果 low <= high:
            交换 arr[low] 和 arr[high]
        否则:
            中断循环
            结束循环

    交换 arr[left] 和 arr[high]
    返回 high

函数 select_kth(list, left, right, k):
    f = left
    r = right
    parti = partition(list, f, r)
    relative_parti = parti - left + 1
    如果 relative_parti == k:
```

```
        返回 list[parti]
    否则如果 relative_parti > k:
        返回 select_kth(list, f, parti - 1, k)
    否则如果 relative_parti < k:
        new_k = k - relative_parti
        返回 select_kth(list, parti + 1, right, new_k)

函数 checkMain_qs(list, length):
    如果 length 为奇数:
        ifMain = select_kth(list, 0, length - 1, (length + 1) / 2)
    否则:
        ifMain = select_kth(list, 0, length - 1, length / 2 + 1)

    count = 0
    对于 i 在 list 中循环:
        如果 i == ifMain:
            count += 1

    如果 count > length / 2:
        返回 True
    否则:
        返回 False
```

3 Question 2-10

3.1 问题描述

若在习题 2-9 中，数组T中元素不存在序关系，只能测试任意两个元素是否相等，试设计一个有效算法确定T是否有一主元素算法的计算复杂性应为。$O(n \log n)$更进一步，能找到一个线性时间算法吗?

3.2 实现算法1

首先，计算需要检查的次数阈值 `check`，即列表长度的一半。然后，创建一个哈希表 `hmap` 来记录每个元素出现的次数。接着，遍历列表，更新哈希表中对应元素的出现次数，并检查是否有元素的出现次数超过了 `check`，如果超过了则返回 `True`，否则返回 `False`。其时间复杂度是

$O(n)$，但是同样也需要$O(n)$的空间复杂度

```
函数 checkMain_hash(list, length):
    check = length / 2
    res = False

    哈希表 hmap: dict = {}
    对于 i 从 0 到 length-1 循环:
        hmap[list[i]] = 0

    对于 i 从 0 到 length-1 循环:
        hmap[list[i]] += 1
        如果 hmap[list[i]] > check:
            res = True
            中断循环
            结束

    返回 res
```

3.3 实现算法2

这段代码实现的是一种基于消除法的查找算法，用于找到可能存在于列表中超过一半次数的元素。其基本思想是通过消除不同的元素，最终剩下的元素即为可能出现次数超过一半的元素。

首先，初始化计数器 count 为 1，以及当前元素 now 为列表的第一个元素。然后，从第二个元素开始遍历列表，如果当前元素与 now 相同，则增加计数器 count；如果不同且计数器大于 0，则减少计数器 count；如果计数器为 0，则更新当前元素 now 为当前列表元素，并将计数器 count 重置为 1。接着，再次遍历列表，统计当前元素 now 出现的次数，如果超过列表长度的一半，则返回 True，否则返回 False。

其时间复杂度是$O(n)$

```
函数 checkMain_minus(list, length):
    count = 1
    now = list[0]
    对于 i 从 1 到 length-1 循环:
        如果 now == list[i]:

            count += 1
```

```
        否则如果 count > 0:
            count -= 1
        否则:
            now = list[i]
            count = 1

    count = 0
    对于 i 在 list 中循环:
        如果 i == now:
            count += 1
    如果 count > length / 2:
        返回 True
    否则:
        返回 False
```