Phân Tích Thuật Toán
Trường Đại Học Khoa Học Tự Nhiên
Nguyễn Thanh Bình

Report lab 7

Nguyễn Quốc Bảo - 18110053

Bài 1. Một mảng A có n phần tử. Ta gọi $x \in A$ là phần tử "lấn chiếm cấp độ 10" nếu như x xuất hiện ít nhất $\left[\frac{n}{10}\right]$ lần trong A.

1. Xây dựng một thuật toán tìm phần tử lấn chiếm cấp 10 trong một mảng.

   Với trường hợp này, ta có thể thực hiện bằng cách dùng map để lưu các phần tử có trong mảng và số lần xuất hiện của chúng.

```
# frequency of elements is more than n/k
def findmajorityNbyK(arr, n, k) :
    global counter_assign
    global counter_compare

    x = n // k

    # map initialization
    freq = {}
    counter_assign += 2


    for i in range(n) :
        counter_compare += 2
        if arr[i] in freq :
            freq[arr[i]] += 1
            counter_assign += 1
        else :
            freq[arr[i]] = 1
            counter_assign += 1
```

   Có thể thấy trong vòng lặp đầu tiên số lần lặp là n (với n là số phần tử của mảng). Khi đó phép so sánh là 2n và phép gán là n.

```
# Store result
ls_result = []
# Traversing the map
for i in freq :
    counter_compare += 2
    # Checking if value of a key-value pair
    # is greater than x (where x=n/k)
    if (freq[i] >= x) :

        # append the key of whose value
        # is greater than x
        ls_result.append(i)

return ls_result
```

Sau khi đã đếm được mọi phần tử trong mảng, thì chỉ cần xét số lần xuất hiện của tập các phần tử có ít nhất là $\frac{n}{10}$ lần thì ta lưu vào kết quả (ls-result). Và vòng lặp này thực hiện tùy vào số lượng của tập phần tử, gọi h là số lượng tập các phần tử, khi đó vòng lặp thực hiện h, phép so sánh là 2h.
Như vậy tổng lần số phép so sánh và số phép gán là $2n + 2h + n = 3n + 2h$. Vì thế có thể nói thuật toán này có độ phức tạp là $O(n)$ vì trượng xấu nhất là h = n thì khi đó tổng số lần thực hiện sẽ là $5n$ và trường hợp tốt nhất là $h = 1$.

Kết quả sau khi chạy như sau:

```
---------------------------------------- Exercise 1 ----------------------------------------
>> Input number of elements array random : 20
initializing random array from 0 to 100:
A =
[78 43 38 43 91 87 68 19  6 70  3  7 26 76 64 47 43  0 38 79]
Elements appear at least n/10 = 2 times:
[43, 38]
When N = 20 , then count_compare: 74 and count_assign: 22
########################################## End Exercise 1 ##########################################
```

2. Liệu có tồn tại thuật toán có độ phức tạp $O(n)$ để tìm ra phần tử kiểu này không?

   Có, như đã nói ở trên thuật trên có độ phức tạp là $O(n)$ có thể tìm được các phần tử có số lần xuất hiện ít nhất $n/10$ lần.

Bài 2. Cho mảng A và B đã được sắp xếp theo thứ tự tăng dần và có n phần tử. Hãy xây dựng 1 thuật toán khả thi để merge hai mảng A và B thành mảng có 2n phần tử đã được xấp xếp theo thứ tự tăng dần

Ta có 2 mảng A và B đã được sắp xếp tăng dần, để thực hiện merge 2 mảng trên và sau khi merge cũng được sắp xếp ta có thể thực hiện theo cách sau:

1. Tạo mảng thứ 3 có độ dài là 2 mảng A và B cộng lại.
2. Duyệt mảng đồng thời 2 mảng A và B, chọn các phần tử nhỏ hơn hiện tại của mảng A và B , sao chép phần tử này vào mảng thứ 3, tiếp tục tăng dần vị trí

.

```python
def mergeArrays(arr1, arr2, n1, n2):
    global counter_assign
    global counter_compare

    arr3 = [None] * (n1 + n2)
    i = 0
    j = 0
    k = 0

    counter_assign += 4

    # Traverse both array
    while i < n1 and j < n2:
        counter_compare += 2

        # Check if current element of first array is smaller
        # than current element of second array.
        # If yes, store first array element and increment first array index.
        # Otherwise do same with second array
        if arr1[i] < arr2[j]:
            arr3[k] = arr1[i]
            k = k + 1
            i = i + 1
            counter_assign += 3
        else:
            arr3[k] = arr2[j]
            k = k + 1
            j = j + 1
            counter_assign += 3
```

3. Nếu còn phần tử trong A hoặc B thì sao chép chúng vào mảng thứ 3.

```
# Store remaining elements
# of first array
while i < n1:
    counter_compare += 1
    counter_assign += 3

    arr3[k] = arr1[i]
    k = k + 1
    i = i + 1

# Store remaining elements
# of second array
while j < n2:
    counter_compare += 1
    counter_assign += 3

    arr3[k] = arr2[j]
    k = k + 1
    j = j + 1

return arr3
```

Ý tưởng này giống với thuật toán mergesort. Vì phải duyệt cả hai mảng nên thuật toán có độ phức tạp là $O(n1 + n2)$ với n1 và n2 là số lượng phần tử của mảng A và B ban đầu nhưng do trong trường hợp này 2 mảng có cùng độ dài mảng là n nên độ phức tạp của thuật toán trên sẽ là $O(2n) \Longrightarrow O(n)$.

Kết quả sau khi chay như sau:

```
------------------------------------ Exercise 2 ------------------------------------
>> Input number of elements array random : 20
initializing random array from 0 to 100:
A =
[5, 13, 16, 22, 24, 25, 28, 29, 34, 35, 38, 41, 53, 56, 67, 81, 91, 93, 97, 99]
B =
[11, 17, 18, 27, 29, 35, 40, 41, 46, 57, 58, 62, 71, 76, 78, 82, 87, 90, 92, 99]

Array after merging A and B:
[ 5, 11, 13, 16, 17, 18, 22, 24, 25, 27, 28, 29, 29, 34, 35, 35, 38, 40, 41, 41,
 46, 53, 56, 57, 58, 62, 67, 71, 76, 78, 81, 82, 87, 90, 91, 92, 93, 97, 99, 99 ]

When total N = 40 , then count_compare: 79 and count_assign: 124
###################################### End Exercise 2 ######################################
```