Assignment 1 of CISC 3000

ZHANG HUAKANG

March 3, 2022

1

I use a Hasp Map to count each number in A and B, and the number that the count is 1 is what we want find.

Algorithm 1:

```
H is a hash map.
for i ∈ A do
    if i ∈ H then
        H_i+=1
    end
    else
        H_i=1
    end
end
for i ∈ B do
    if i ∈ H then
        H_i+=1
    end
    else
        H_i=1
    end
end
for i ∈ H do
    if H_i = 1 then
        Output: i ∈ (A ∪ B) \ (A ∩ B)
    end
end
```

This algorithm is the count of each number in A or B is 1. We will prove that if the count of number is 1, this number must in $(A \cup B) \setminus (A \cap B)$

Proof. Since that elements in A have different values and elements in B also have different values, each element E in A or B will only show one time in its array, denoted as $No_A(E) = 1$ or $No_B(E) = 1$. If we put the elements in two array together into array C which allows duplicate, the number of each elements $No_C(E)$ will have two case. For element $E \in A \cup B$,

Case 1 If $E \in A$ and $E \notin B$, then $No_C(E) = 1$.

Case 2 If $E \notin A$ and $E \in B$, then $No_C(E) = 1$.

Case 3 If $E \in A$ and $E \in B$, then $No_C(E) = 2$.

So, if $No_C(E) = 1$, then ($E \in A$ and $E \notin B$) or ($E \notin A$ and $E \in B$), *i.e.*, if $E \in (A \cup B) \setminus (A \cap B)$ □

Complexity The hash operation complexity is $O(1)$

$$\begin{aligned} T(|A| + |B|) =& |A| \times O(1) + |B| \times O(1) + |(A \cup B) \setminus (A \cap B)| \times O(1) \\ =& |A| + |B| + |(A \cup B) \setminus (A \cap B)| \\ <& |A| + |B| + |A| + |B| \\ =& 2(|A| + |B|) \\ =& O(|A| + |B|) \end{aligned} \tag{1}$$

2

a

Algorithm 2:

$p_L = 1$, $p_R = 1$, $count = 0$.
while $p_L \leq |L|$ *or* $p_R \leq |R|$ **do**
  if $L_{p_L} > R_{p_R}$ **then**
    $count += 1$
    $p_R += 1$
  end
  else
    $p_L += 1$
  end
end
Output: *count*

Complexity We have a pointer for each array. **For each while-loop, there must be only one poniter can move.** They both start from the begining of the array, and stop when both of them reach to the end of the array. p_L moves $|L|$ times, and p_R moves $|R|$ times. For the whole while-loop, it will be executed

$|L| + |R|$ times. We know that $count \in [0, |L| + |R|]$

$$
\begin{aligned}
T(|L| + |R|) =& 2 \times count + (|L| + |R| - count) \\
=& count + |L| + |R| \\
\leq& |L| + |R| + |L| + |R| \\
=& 2(|L| + |R|) \\
=& O(|L| + |R|)
\end{aligned}
\tag{2}
$$

b

```
function merge(A : array, p : int, q : int, r : int)
    n_1 = q - p + 1
    n_2 = r - q
    let L[1..n_1 + 1] and R[1..n_2 + 1]
    for i = 1 to n_1 do
        L[i] = A[p + i - 1]
    end
    for i = 1 to n_2 do
        R[i] = A[q + j]
    end
    L[n_1 + 1] = R[n_2 + 1] = ∞
    i = j = 1
    for k = p to r do
        if L[i] ≤ R[j] then
            A[k] = L[i]
            i+ = 1
        end
        else
            A[k] = R[i]
            j+ = 1
        end
    end
end
function merge-sort(A : array, p : int, r : int)
    if p < r then
        q = ⌊(p + r)/2⌋
        merge-sort(A,p,q)
        merge-sort(A,q+1,r)
        // Count the inversion between A[p, q] and A[q + 1, r],
         O(q - p + 1)
        count-inversion(A,p,q,r)
        merge(A,p,q,r)
    end
end
```

Complexity It is easy to find that

$$\begin{aligned} T_{merge}(p,r) =& r-p+1 \\ =& O(r-p) \end{aligned} \tag{3}$$

$$\begin{aligned} T(n) =& 2\times T(\frac{n}{2}) + T_{merge}(1,n) + cn \\ =& 2\times T(\frac{n}{2}) + c'n \\ =& 2\log n \times c'n + c'n \\ =& O(n\log n) \end{aligned} \tag{4}$$

3

Proof. $T(1), T(2), T(3) \le c = O(1)$ and

$$T(n) \le T(\frac{n}{4}) + T(\frac{3}{4}) + cn \tag{5}$$

when $n \ge 4$. Thus,

$$\begin{aligned} T(4) \le& T(1) + T(3) + cn \\ =& 2c + 4c \\ =& 6c \\ \le& \frac{6c}{4\log 4} 4\log 4 \\ =& O(n\log n) \end{aligned} \tag{6}$$

Suppose that $T(k) = O(k\log k), k \ge 4$, we have

$$T(k+1) \le T(\frac{k+1}{4}) + T(\frac{3(k+1)}{4}) + c(k+1) \tag{7}$$

$\square$