论题 2-14 作业

姓名：陈劭源　　　　学号：161240004

1 [TC] Problem 21.1-2

"If": we use mathematical induction to prove this. For base step, a and a are in the same set, and a and a are in the same connected component. For induction step, if S is a set, then S is either a set with only one element, or a set which is a union of two disjoint sets, say S_1 and S_2, connected by edge (u, v) where $u \in S_1$ and $v \in S_2$. In the latter case, for every two elements x and y in S, if both x and y are in S_1 or S_2, then x and y are in the same connected component by induction hypothesis. Otherwise, assume $x \in S_1$, $y \in S_2$, then x has a path to u, v has a path to v, and there exists edge (u, v), thus x and y are connected. Therefore, if two elements are in the same set, they are in the same connected component.

"Only if": if a and b are in the same connected component, then there exists a path from a to b. After the procedure executed, all edges in the path have been processed and all these vertices have been united, thus a and b are in the same set.

2 [TC] Problem 21.1-3

There are $|E|$ edges in the graph, and for every edge, FIND-SET is called twice, thus FIND-SET is called $2|E|$ times in all.

The edges, where UNION is performed, constitute the spanning trees of the connected components. For the ith connected component, assume there are $|V_i|$ vertices, then its spanning tree has $|V_i| - 1$ edges. Therefore, there are $|V| - k$ edges in the spanning trees, so UNION is called $|V| - k$ times in all.

3 [TC] Problem 21.2-1

MAKE-SET(x)

let S be new linked list
$S.head = x$
$S.tail = x$
$S.weight = 1$
$x.root = S$
$x.next =$ NIL

FIND-SET(x)

return $x.root.head$

```
UNION(x, y)
if x.weight < y.weight
    swap x and y
t = y.head
while t ≠ NIL
    t.root = x
    t = t.next
x.tail.next = y.head
x.tail = y.tail
x.weight = x.weight + y.weight
```

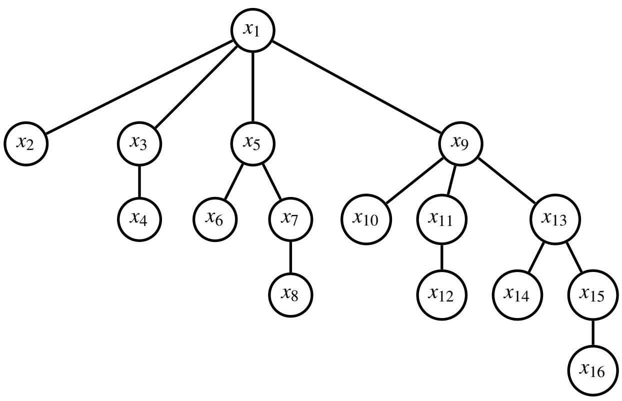
4 [TC] Problem 21.2-3

It is obvious that MAKE-SET and FIND-SET take an amortized running time of $O(1)$. We have proved in Theorem 21.1, that we perform at most $n-1$ UNION operations over all, and the total time spent on UNION is $O(n \lg n)$. Thus the amortized running time of UNION is $O(\lg n)$.

5 [TC] Problem 21.2-6

```
UNION(x, y)
    // if weighted-union heuristic is used
if x.weight < y.weight
    swap x and y
t = y.head
while t.next ≠ NIL
    t.root = x
    t = t.next
t.root = x
t.next = x.head
x.head = y.head
```

6 [TC] Problem 21.3-1



The answers returned by the FIND-SET operations is x_1.

7 [TC] Problem 21.3-2

```
FIND-SET(x)
y = x
while y.p ≠ y
    y = y.p
z = x
while z ≠ y
    x = z.p
    z.p = y
    z = x
return y
```

8 [TC] Problem 21.3-3

Assume $A[0 \cdots n-1]$ is an array of elements. The sequence is given by the following procedure.

```
1  for i = 0 to n − 1
2      MAKE-SET(x_i)
3  j = 2
4  while j < n
5      for i = 0 to n by j
6          if i + j/2 < n
7              UNION(x_i, x_{i+j/2})
8      j = 2j
9  t = 2^{⌊lg n⌋} − 1
10 for i = 1 to m − 2n + 1
11     FIND-SET(x_t)
```

Each iteration of **while** (except the last one if n is not a power of 2) in line 4-8 increments the depth of x_t by 1. So the depth of x_t is $\lfloor \lg n \rfloor$ at last. There are n calls to MAKE-SET and $n-1$ calls to UNION, each taking $\Omega(1)$ time. The $m-2n+1$ calls to FIND-SET take $\Omega(\lg n)$ time each. The total running time is $\Omega(2n-1+(m-2n+1)\lg n)$, and if $m = \omega(n)$, it is $\Omega(m \lg n)$.

9 [TC] Problem 21-1

a. $\{4, 3, 2, 6, 8, 1\}$

b. We use the following loop invariant to prove the correctness:

> Before each iteration of **for** loop, for every j, $extracted[j]$ is either empty, or filled with correct value; if it is empty, then：
>
> (1) the correct value of $extracted[j]$ is greater than or equal to i, or the set is empty when the corresponding EXTRACT-MIN is called;

(2) K_j exists and $K_j = \cap_{i=k}^{j} I_j$, where k is the minimum of l such that for every i between l and j, either $i = j$ or $extracted[i]$ has been filled with correct value. (this is also true for $j = m+1$)

Initialization: Prior to the first iteration, all elements of *extracted* is empty, and both (1) and (2) are correct for every j.

Maintenance: During the iteration, if $j = m+1$, by loop invariant (2), there is no unprocessed EXTRACT-MIN after i in the original sequence, and the loop invariant holds. If $j \neq m+1$, there exist some unprocessed EXTRACT-MINs after i in the original sequence, among which the first one to appear is j, according to loop invariant (2). By loop invariant (1), $extracted[j]$ must be i. Line 7 maintains the loop invariant (2). Therefore, the loop invariant still holds after each iteration.

Termination: When the loop terminates, $i = n+1$. For every j, if $extracted[j]$ is empty, the set must be empty when the corresponding EXTRACT-MIN is called, because the correct value of $extracted[j]$ can't be greater than or equal to $n+1$. Thus, the array *extracted* returned by OFF-LINE-MINIMUM is correct.

c. To find the smallest value greater than j for which set K_l exists quickly, we shall maintain a linked list of the sets. When K_j is destroyed, the set should be deleted from the list. It takes $O(m)$ time to build the list, and $O(1)$ time to delete an element. There are n itertaions, and every element of *extracted* is filled at most once, which takes an amortized running time of $O(\alpha(n))$, if both union by rank and path compression are used. Thus, the total running time if $O(n + m\alpha(n))$.