Problem Solving Homework(Week 14)

161180162 Xu Zhiming

May 31, 2017

TC

21.1-2

Proof. **Sufficiency:** If two vertices(u and v) are in the same component, then in line 4 and 5 in CONNECTED-COMPONENTS(G), they are put in the same set by method UNION(u,v).
Necessity: In CONNECTED-COMPONENT(G), method UNION is only used in line 5, when u and v are connected by an edge, that's to say, u and v are in the same set if u and v are in the same component. □

21.1-3

Assume that at first we have $|V|$ components and they are not connected. At last, we have only k components. Performing UNION(G) once can reduce the number of components by 1. So the total number of performing UNION(G) is $|V| - k$. As for FIND-SET(u,v), it is performed $2 \times |E|$ times, because for each edge uv, FIND-SET(u,v) will be performed twice.

21.2-1

```
//Linked-list and weighted union
MAKE-SET(x[i])
let S[i] be a new linked-list
S[i].head=x[i]
S[i].tail=x[i]
x[i].next=NIL
x[i].p=S[i]
S[i].length=1
FIND-SET(x)
return x.p.head
UNION(x,y)
if x.p.length>y.p.length
        tmp_x=x.p.tail
        tmp_y=y.p.head
        do
                tmp_x.next=tmp_y
                tmp_y.p=tmp_x.p
                tmp_x.p.tail=tmp_y
                tmp_x=tmp_y
                tmp_y=tmp_y.next
        when
                tmp_y!=NIL
        x.length+=y.length
else
        tmp_y=y.p.tail
        tmp_x=x.p.head
        do
                tmp_y.next=tmp_x
                tmp_x.p=tmp_y.p
                tmp_y.p.tail=tmp_x
                tmp_y=tmp_x
```

```
                tmp_x=tmp_x.next
        when
                        tmp_x!=NIL
        y.length+=x.length
```

21.2-3
For MAKE-SET and FIND-SET, m times performances require $O(m)$ time individually. So the amortized time bound is $O(1)$.
For UNION, every time it is performed, the size of a set increases to at least twice as large as before. So, union n one-element set requires at most $O(nlgn)$ times modifications to the elements in the smaller set. So each UNION running amortized running time is $O(lgn)$.
21.2-6
Let head has two member pointers: "first" and "second", "first" pointing to a list, "second" points to the other.
21.3-1
21.3-2

```
NONRECURSIVE-FIND-SET(x)
xx=x
while xx!=xx.p
        xx=xx.p
while(x!=x.p)
        tmp=x.p
        x.p=xx
        x=tmp
return x.p
```

21.3-3

```
for i=1 to n
        MAKE-SET(x[i])    //n
for i=1 to n
        UNION(x[i],x[1])//n
for i=1 to m-(2*n-1)
        FIND-SET(x[n])    //(m-2n+1)lgn
```

Overall: $\Omega(2n + (m - 2n + 1)lgn) = \Omega(mlgn)$
21-1
a: $extrace[6] = 4, 3, 2, 6, 8, 1$.
b:

Proof. **Loop invariant:** Every time before the for-loop starts, the set that contains i has the index that indicates i's position in array *extracted*.
Initialize: When $i = 1$, i is the minimum element globally, so the set that contains 1 will eject it first.
Maintenance: When the k^{th} loop is over, the $k + 1^{th}$ loop starts. Assume that $k \in K_s$, then $K_r = K_s \cup K_r(K_r$ exits and r is the smallest number larger than s). Than if $k + 1$ originally belongs to K_r, than it will be ejected from it. $\square$

c:

```
OFF-LINE-MINIMUM(m,n)    //Disjoint-set data structure
for i=1 to n
        if      FIND-SET(i) belongs to K[j]
                if      j!=m+1
                        extracted[j]=i
                        let l be the smallest value greater than j for which set K[l] exits
                        UNION(K[l],K[j])
return extracted
```