3-6 Union-Find

Jun Ma

majun@nju.edu.cn

October 27, 2020

TC 21.1-2

Show that after all edges are processed by CONNECTED - COMPONENTS, two vertices are in the same connected component if and only if they are in the same set.

```
CONNECTED-COMPONENTS(G)
    X ← ∅
1   for each vertex v ∈ G.V
2       MAKE-SET(v)
3   for each edge (u, v) ∈ G.E
4       if FIND-SET(u) ≠ FIND-SET(v)
5           UNION(u, v)
            X ← X + {(u, v)}
```

* Each set of the forest obtained so far corresponds to one component of the graph $G'(V, X)$

TC 21.1-3

During the execution of CONNECTED-COMPONENTS on an undirected graph $G = (V, E)$ with k connected components, how many times is FIND-SET called? How many times is UNION called? Express your answers in terms of $|V|$, $|E|$, and k.

- How many times is FIND-SET called?
  - $2|E|$
- How many times is UNION called?
  - $|V| - k$

TC 21.2-1

Write pseudocode for MAKE-SET, FIND-SET, and UNION using the linked-list representation and the weighted-union heuristic. Make sure to specify the attributes that you assume for set objects and list objects.

```
1: procedure MAKE-SET(x)
2:     let L be a new list
3:     L.head = x
4:     L.tail = x
5:     L.size = 1
6:     x.next = L
7:     x.anc = L
8: end procedure
```

```
1: function FIND-SET(x)
2:     return x.anc.head
3: end function
```

```
1: procedure UNION(u, v)
2:     L_1 = u.anc
3:     L_2 = v.anc
4:     if L_1.size < L_2.size then
5:         Swap L_1, L_2
6:     end if
7:     x = L_2.head
8:     x.anc = L_1
9:     L_1.tail.next = L_2.head
10:    while true do
11:        x.anc = L_1
12:        if x = L_2.tail then
13:            break
14:        end if
15:        x = x.next
16:    end while
17:    L_1.size = L_1.size + L_2.size
18:    L_1.tail = L_2.tail
19:    delete L_2
20: end procedure
```

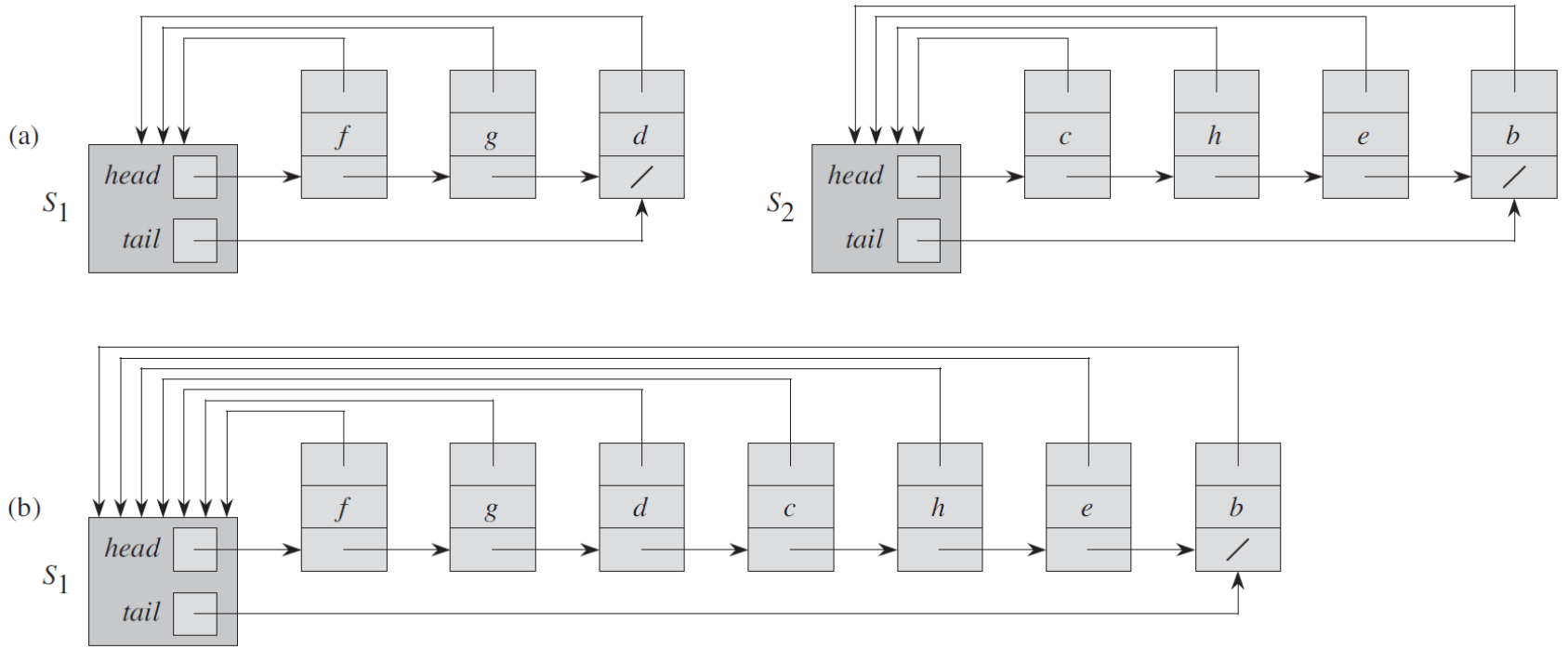
TC 21.2-3

Adapt the aggregate proof of Theorem 21.1 to obtain amortized time bounds of $O(1)$ for MAKE-SET and FIND-SET and $O(\lg n)$ for UNION using the linked-list representation and the weighted-union heuristic.

- Trivial.

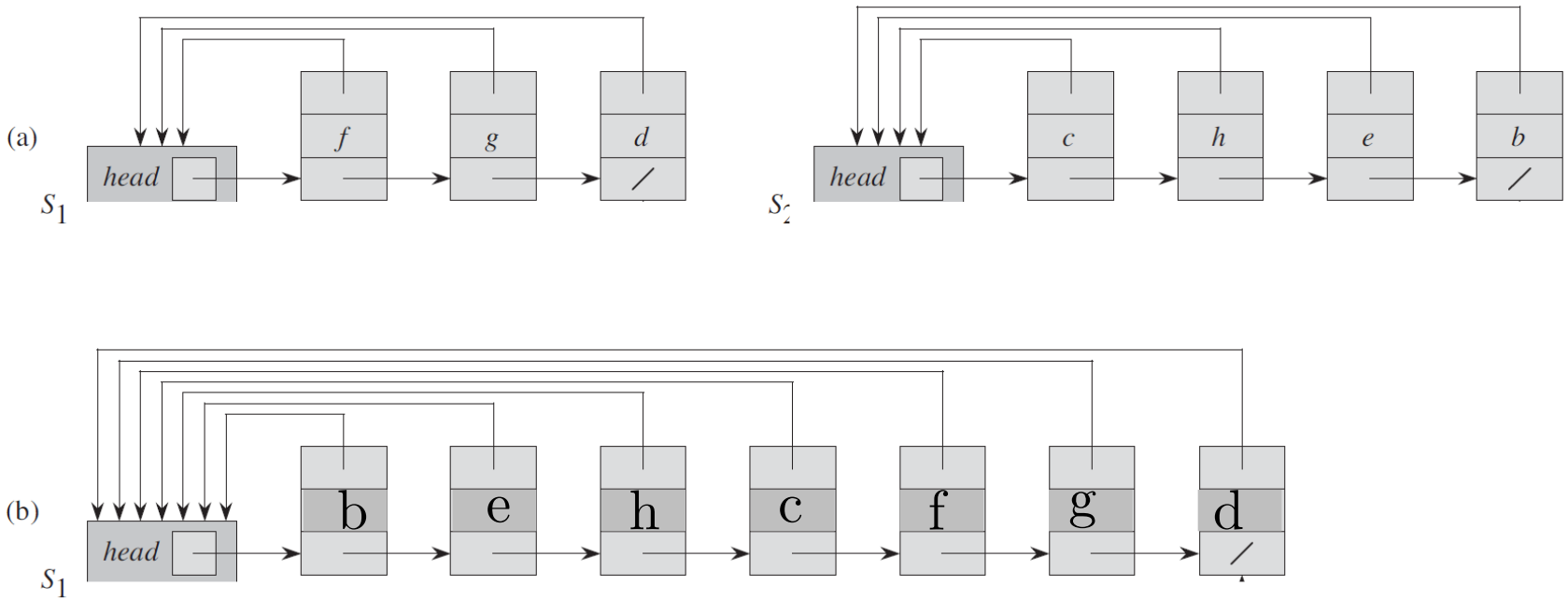
TC 21.2-6

Suggest a simple change to the UNION procedure for the linked-list representation that removes the need to keep the tail pointer to the last object in each list. Whether or not the weighted-union heuristic is used, your change should not change the asymptotic running time of the UNION procedure.



TC 21.2-6

Suggest a simple change to the UNION procedure for the linked-list representation that removes the need to keep the tail pointer to the last object in each list. Whether or not the weighted-union heuristic is used, your change should not change the asymptotic running time of the UNION procedure.



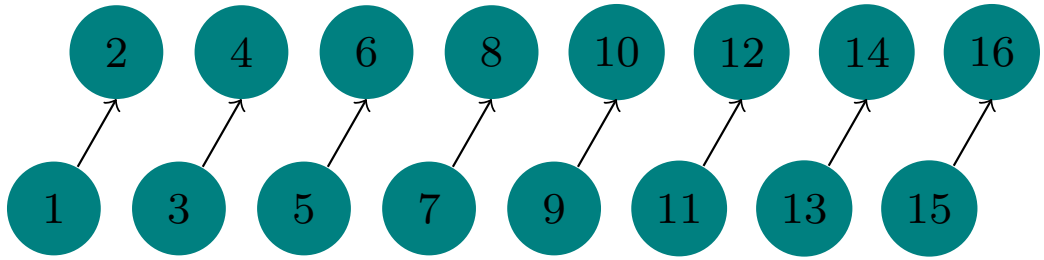
TC 21.3-1

Redo Exercise 21.2-2 using a disjoint-set forest with union by rank and path compression.

21.2-2

Show the data structure that results and the answers returned by the FIND-SET operations in the following program. Use the linked-list representation with the weighted-union heuristic.

```
1   for i = 1 to 16
2       MAKE-SET(x_i)
3   for i = 1 to 15 by 2
4       UNION(x_i, x_{i+1})
5   for i = 1 to 13 by 4
6       UNION(x_i, x_{i+2})
7   UNION(x_1, x_5)
8   UNION(x_11, x_13)
9   UNION(x_1, x_10)
10  FIND-SET(x_2)
11  FIND-SET(x_9)
```



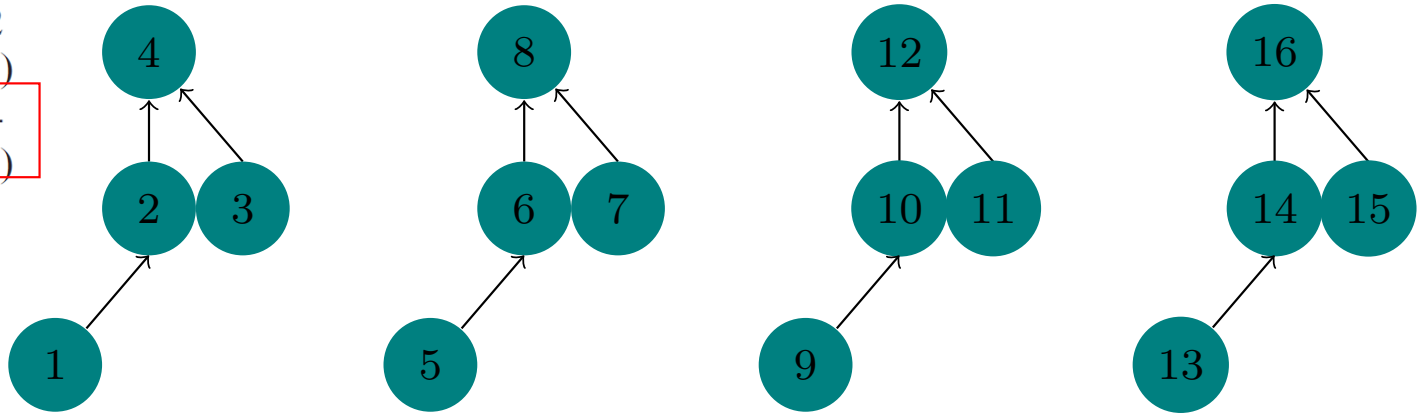
TC 21.3-1

Redo Exercise 21.2-2 using a disjoint-set forest with union by rank and path compression.

21.2-2

Show the data structure that results and the answers returned by the FIND-SET operations in the following program. Use the linked-list representation with the weighted-union heuristic.

```
1   for i = 1 to 16
2       MAKE-SET(x_i)
3   for i = 1 to 15 by 2
4       UNION(x_i, x_{i+1})
5   for i = 1 to 13 by 4
6       UNION(x_i, x_{i+2})
7   UNION(x_1, x_5)
8   UNION(x_11, x_13)
9   UNION(x_1, x_10)
10  FIND-SET(x_2)
11  FIND-SET(x_9)
```



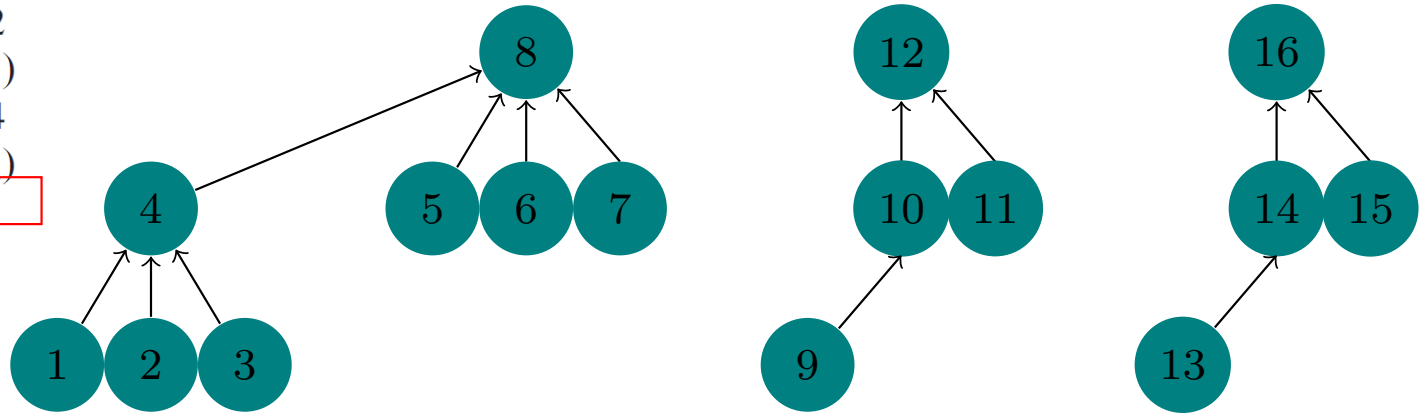
TC 21.3-1

Redo Exercise 21.2-2 using a disjoint-set forest with union by rank and path compression.

21.2-2

Show the data structure that results and the answers returned by the FIND-SET operations in the following program. Use the linked-list representation with the weighted-union heuristic.

```
1   for i = 1 to 16
2       MAKE-SET(x_i)
3   for i = 1 to 15 by 2
4       UNION(x_i, x_{i+1})
5   for i = 1 to 13 by 4
6       UNION(x_i, x_{i+2})
7   UNION(x_1, x_5)
8   UNION(x_11, x_13)
9   UNION(x_1, x_10)
10  FIND-SET(x_2)
11  FIND-SET(x_9)
```



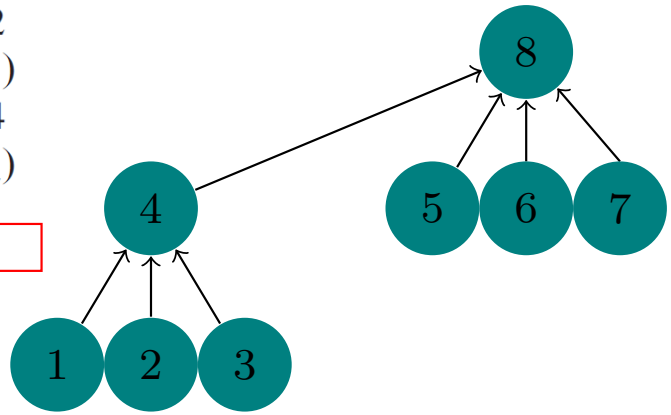
TC 21.3-1

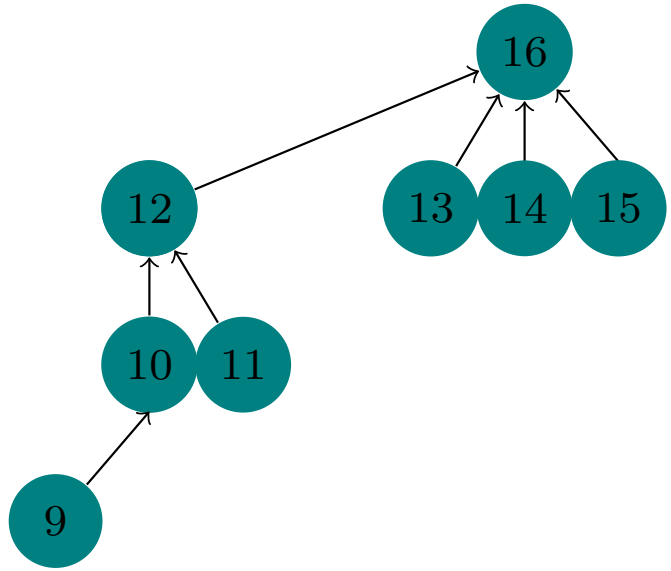
Redo Exercise 21.2-2 using a disjoint-set forest with union by rank and path compression.

21.2-2

Show the data structure that results and the answers returned by the FIND-SET operations in the following program. Use the linked-list representation with the weighted-union heuristic.

```
1   for i = 1 to 16
2       MAKE-SET(x_i)
3   for i = 1 to 15 by 2
4       UNION(x_i, x_{i+1})
5   for i = 1 to 13 by 4
6       UNION(x_i, x_{i+2})
7   UNION(x_1, x_5)
8   UNION(x_11, x_13)
9   UNION(x_1, x_10)
10  FIND-SET(x_2)
11  FIND-SET(x_9)
```





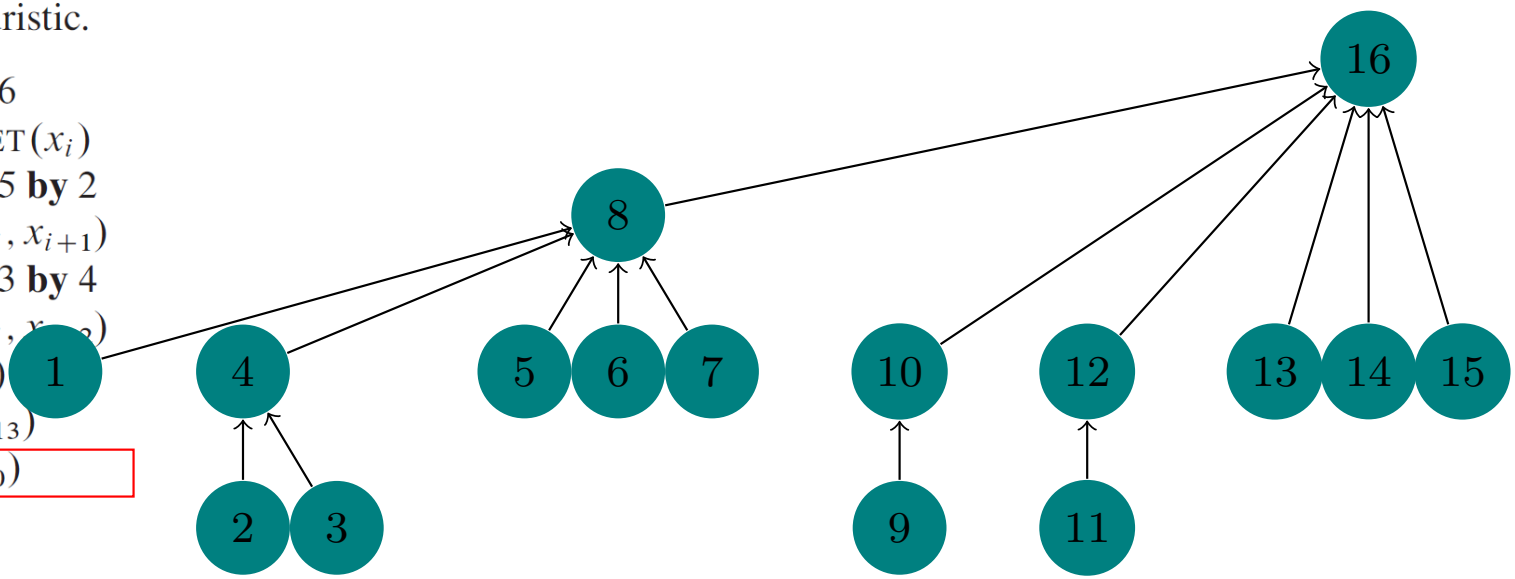
TC 21.3-1

Redo Exercise 21.2-2 using a disjoint-set forest with union by rank and path compression.

21.2-2

Show the data structure that results and the answers returned by the FIND-SET operations in the following program. Use the linked-list representation with the weighted-union heuristic.

```
1   for i = 1 to 16
2       MAKE-SET(x_i)
3   for i = 1 to 15 by 2
4       UNION(x_i, x_{i+1})
5   for i = 1 to 13 by 4
6       UNION(x_i, x_{i+2})
7   UNION(x_1, x_5)
8   UNION(x_11, x_13)
9   UNION(x_1, x_10)
10  FIND-SET(x_2)
11  FIND-SET(x_9)
```



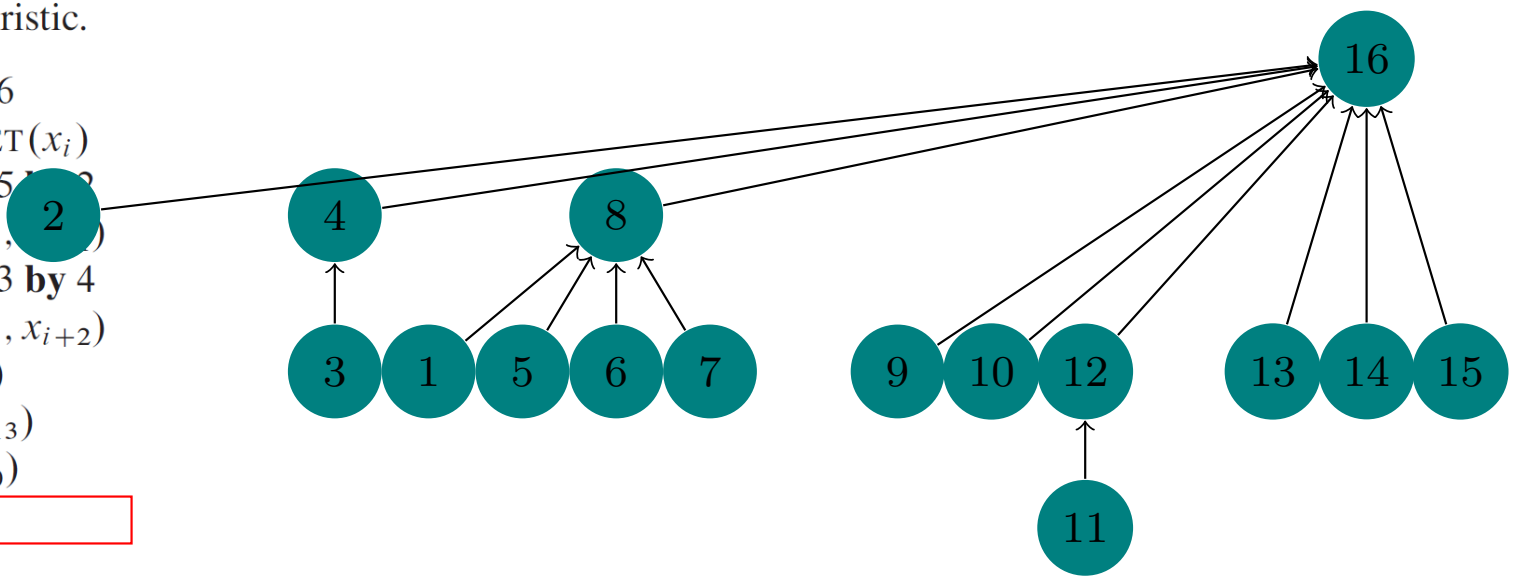
TC 21.3-1

Redo Exercise 21.2-2 using a disjoint-set forest with union by rank and path compression.

21.2-2

Show the data structure that results and the answers returned by the FIND-SET operations in the following program. Use the linked-list representation with the weighted-union heuristic.

```
1   for i = 1 to 16
2       MAKE-SET(x_i)
3   for i = 1 to 15
4       UNION(x_i, x_{i+1})
5   for i = 1 to 13 by 4
6       UNION(x_i, x_{i+2})
7   UNION(x_1, x_5)
8   UNION(x_11, x_13)
9   UNION(x_1, x_10)
10  FIND-SET(x_2)
11  FIND-SET(x_9)
```



TC 21.3-2

Write a nonrecursive version of FIND-SET with path compression.

```
1: function FIND-SET(x)
2:     y = x
3:     while y ≠ fa[y] do
4:         y = fa[y]
5:     end while
6:     while x ≠ fa[x] do
7:         z = x
8:         x = fa[x]
9:         fa[z] = y
10:    end while
11:    return y
12: end function
```

TC 21.3-3

Give a sequence of m MAKE-SET, UNION, and FIND-SET operations, n of whichare MAKE-SET operations, that takes $O(m \lg n)$ time when we use union by rank only.

TC Problem 21-1 (Off-line minimum)

The ***off-line minimum problem*** asks us to maintain a dynamic set T of elements from the domain $\{1, 2, \ldots, n\}$ under the operations INSERT and EXTRACT-MIN. We are given a sequence S of n INSERT and m EXTRACT-MIN calls, where each key in $\{1, 2, \ldots, n\}$ is inserted exactly once. We wish to determine which key is returned by each EXTRACT-MIN call. Specifically, we wish to fill in an array $extracted[1..m]$, where for $i = 1, 2, \ldots, m$, $extracted[i]$ is the key returned by the ith EXTRACT-MIN call. The problem is "off-line" in the sense that we are allowed to process the entire sequence S before determining any of the returned keys.

TC Problem 21-1 (Off-line minimum)

a. In the following instance of the off-line minimum problem, each operation INSERT(i) is represented by the value of i and each EXTRACT-MIN is represented by the letter E:

$4, 8, \mathrm{E}, 3, \mathrm{E}, 9, 2, 6, \mathrm{E}, \mathrm{E}, \mathrm{E}, 1, 7, \mathrm{E}, 5$.

Fill in the correct values in the *extracted* array.

$$[4, 3, 2, 6, 8, 1]$$

TC Problem 21-1 (Off-line minimum)

To develop an algorithm for this problem, we break the sequence S into homogeneous subsequences. That is, we represent S by

$$\mathrm{I}_1, \mathrm{E}, \mathrm{I}_2, \mathrm{E}, \mathrm{I}_3, \ldots, \mathrm{I}_m, \mathrm{E}, \mathrm{I}_{m+1} ,$$

where each E represents a single EXTRACT-MIN call and each I_j represents a (possibly empty) sequence of INSERT calls. For each subsequence I_j, we initially place the keys inserted by these operations into a set K_j, which is empty if I_j is empty. We then do the following:

```
OFF-LINE-MINIMUM(m, n)
1  for i = 1 to n
2      determine j such that i ∈ K_j
3      if j ≠ m + 1
4          extracted[j] = i
5          let l be the smallest value greater than j
               for which set K_l exists
6          K_l = K_j ∪ K_l, destroying K_j
7  return extracted
```

TC Problem 21-1 (Off-line minimum)

Argue that the array extracted returned by OFF-LINE-MINIMUM is correct.

OFF-LINE-MINIMUM(m, n)

```
1  for i = 1 to n
2      determine j such that i ∈ K_j
3      if j ≠ m + 1
4          extracted[j] = i
5          let l be the smallest value greater than j
               for which set K_l exists
6          K_l = K_j ∪ K_l, destroying K_j
7  return extracted
```


Thank
You!