Week 4: Assignment 4

Your last recorded submission was on 2023-02-22, 01:15 IST Due date: 2023-02-22, 23:59 IST.

1) A connected component or simply component of an undirected graph is a subgraph in which each pair of nodes is connected with each other via a path.

Consider an undirected graph with `11` nodes labeled from `0` to `10` and a list of undirected edges (each edge is a pair of nodes) is given below:-

`[(0,1),(1,3),(0,3),(5,8),(6,10),(7,1),(7,0),(7,3),(9,4),(9,2),(4,2)]`

The number of connected components is ___.

4

1 point

Question 2 & 3

Consider the following Pseudocode for `findSet` and `unionSet` operation for Disjoint Set Union data structure:

```
parent[n]  // A array of size n such that parent[i] = i
findSet(u):
    If(parent[u] == u):
        Return u
    Return findSet(parent[u])

unionSet(u, v):
    u = findSet(u)
    v = findSet(v)
    If(u != v):
        parent[v] = u
```

2) What would be the worst-case time complexity for given `findSet()` operation? ***1 point***

- ○ $O(n\ log\ n)$
- ○ $O(log\ n)$
- [✓ $O(n)$]
- ○ $O(1)$

3) What would be the worst-case time complexity for given `unionSet()` operation? ***1 point***

- ○ $O(1)$
- ○ $O(log\ n)$
- [✓ $O(n)$]
- ○ $O(n\ log\ n)$

4) Consider the following updated pseudocode for `findSet` with path compression. ***1 point***

```
parent[n]  // A array of size n such that parent[i]=i
findSet(u):
    If(parent[u]==u):
        Return u
    Return parent[u] = findSet(parent[u])
```

Suppose, for an instance, parent array `parent` looks like as following for 8 vertices:-

v	1	2	3	4	5	6	7	8
parent[v]	1	1	1	2	4	4	4	6

What would be the state of the parent array `parent` after performing the `findSet(8)` operation?

○

v	1	2	3	4	5	6	7	8
parent[v]	1	1	1	1	4	4	4	1

[✓ ●]

v	1	2	3	4	5	6	7	8
parent[v]	1	1	1	1	4	1	4	1

○

v	1	2	3	4	5	6	7	8
parent[v]	1	1	1	1	1	1	1	1

○

v	1	2	3	4	5	6	7	8
parent[v]	1	1	1	1	1	4	4	1

5) Consider the following pseudocode for `unionSet` with the union by rank. ***1 point***

```
Rank[n] // A array of size n such that rank[i]=0.
unionSet(u, v):
    u = findSet(u)
    v = findSet(v)
    If(u!=v):
        ## statement_block
```

Which of the following is correct `statement_block` inside the `if` condition to complete the pseudocode for `unionSet` with the union by rank?

[✓]

```
If(Rank[u] < Rank[v]):
    Swap(u, v)
parent[v] = u
If(Rank[u] == Rank[v]):
    Rank[u] = Rank[u] + 1
```

○

```
If(Rank[u] > Rank[v]):
    Swap(u, v)
parent[v] = u
If(Rank[u] == Rank[v]):
    Rank[u] = Rank[u] + Rank[v]
```

○

```
If(Rank[u] < Rank[v]):
    Rank[v] = Rank[v] + 1
    Swap(u, v)
parent[v] = u
If(Rank[u] == Rank[v]):
    Rank[u] = Rank[u] + Rank[v]
```

○

```
If(Rank[u] < Rank[v]):
    Rank[v] = Rank[v] + 1
    parent[v] = u
If(Rank[u] == Rank[v]):
    Rank[u] = Rank[u] + 1
```

6) Recall Destroying Array problem discussed in class, following is the modified version of the same problem:

You are given an array consisting of n non-negative integers . You are going to destroy integers $a_1, a_2, \ldots, a_n$ in the array one by one. Thus, you are given the permutation of integers from 1 to n defining the order elements of the array are destroyed. After each element is destroyed, you have to find out the segment of the array, such that it contains no destroyed elements and the sum of its elements is **minimum** possible. The sum of elements in the empty segment is considered to be 0.

Consider the following input

```
n = 13  #size of array
A = 12 9 17 5 0 6 5 1 3 1 17 14 2  #elements of array
P = 3 7 5 8 12 9 13 11 6 1 10 2 4  #permutation for destroyed array
```

Find the sum of all of the expected outputs.

64

1 point

7) Consider a simple undirected graph G = (V, E). We want to determine if there is a cycle in the graph. A cycle in a graph is defined as a path in G where start and end points or vertices coincide and no edge is visited twice (not allowed to retrace the path backward mid-way). You are trying to come up with an algorithm that involves Disjoint Set Union to solve the problem. ***1 point***

Consider the following algorithm using Disjoint Set Union :

```
def find_cycle(G):
    V = G[V] #extract all the vertices (0 to |V|-1)
    E = G[E] #extract all the edges
    parent = [-1]*length(V) #declaring parent array
    parent[i] = i #initialization for each vertex i in G
    is_cycle = false
    for edge in E:
        u,v = vertices connected by edge
        if findSet(u)==findSet(v):
            do_something_1
        else:
            do_something_2
    return is_cycle
```

Note: → The function `findSet(i)` returns the highest hierarchical parent (the leader vertex of the component to which vertex `i` belongs).

→ `unionSet(u,v)` assigns `parent[u] = findSet(v)`

Select more appropriate code lines to replace *do_something_1* and *do_something_2* in the given algorithm.

○
do_something_1 => `return is_cycle`
do_something_2 => `parent[u] = parent[v]`

○
do_something_1 => `return true`
do_something_2 => `return false`

[✓ ●
do_something_1 => `is_cycle = true`]

do_something_2 => `unionSet(u,v)`

○

do_something_1 => `is_cycle = false`

do_something_2 => `unionSet(u,v)`

8) As part of the new Digital India initiative, the government has decided to connect all capital cities via optical fiber cables to improve the network speed. The cost of this higher quality low latency cable is quite high, and hence the aim of this project is to plan the network in a way such that it minimizes the total length of the cable required to make all the capital cities connected. ***1 point***

Suppose you are given the distance between each pair of capital cities. Consider that we are using Disjoint Set Union in an algorithm where the function `findSet(i)` returns the highest hierarchical parent (the leader vertex of the component to which vertex `i` belongs) and `unionSet(u,v)` assigns `parent[u] = findSet(v)`

To returns the minimum length of cable required to connect all the capital cities, which of the following algorithm is correct?

○

```
E = G[E] #extract all the edges in array [(u,v,d),...] where d is distance
between cities u and v
def find_min_length(E):
    total_length = 0
    for each edge(u,v,d) in E:
        if findSet(u) != findSet(v):
            total_length += d
            unionSet(u,v)
    return total_length
```

[✓]

```
E = G[E] #extract all the edges in array [(u,v,d),...] where d is distance
between cities u and v
def find_min_length(E):
    sort(E)# sort the edges according to their weights in ascending order
    total_length = 0
    for each edge(u,v,d) in E:
        if findSet(u) != findSet(v):
            total_length += d
            unionSet(u,v)
    return total_length
```

○

```
E = G[E] #extract all the edges in array [(u,v,d),...] where d is distance between cities u and v
def find_min_length(E):
    sort(E)# sort the edges according to their weights in ascending order
    total_length = 0
    for each edge(u,v,d) in E:
        if findSet(u) != findSet(v):
            unionSet(u,v)
        else:
            total_length += d
    return total_length
```

○

```
E = G[E] #extract all the edges in array [(u,v,d),...] where d is distance between cities u and v
def find_min_length(E):
    sort(E)# sort the edges according to their weights in descending order
    total_length = 0
    for each edge(u,v,d) in E:
        if findSet(u) != findSet(v):
            unionSet(u,v)
            total_length += d
    return total_length
```

You may submit any number of times before the due date. The final submission will be considered for grading.

Submit Answers