a2

Monthon Kraitheerawut

15 October 2021

1 Question1

Answer
(a) The running time = O(log n)

We can see that f(k) $\leq \log_2(k)$ from below

f(1) = 0
f(2) = 1
f(4) = 2
f(7) = 0
f(8) = 3

Amortized cost:

$\sum_{n=2}^{n} f(k) \leq log_2(\sum_{n=2}^{n} k)$

Divide the whole equation with n

$$\leq log_2(\frac{n(n+1)}{2n})$$
$$log_2(\frac{n+1}{2})$$

.
(b) The running time = O(n)

We can see that f(k) k from below

f(1) = 1
f(2) = 1
f(4) = 4
f(7) = 1
f(8) = 4

f(9) = 9

Amortized cost:

$\sum_{n=1}^{n} f(k) \leq \sum_{n=1}^{n} k$

Divide the equation with n

$$\leq \frac{n(n+1)}{2n}$$
$$\leq \frac{n+1}{2}$$

.

(c) The running time = O(n(log(n)))

(d) The running time = O(n log(n))

(e) The running time = O(log n)

We can see that f(k) $\leq log_2 k$ from below.

f(1) = 1
f(2) = 1
f(4) = 2
f(7) = 1
f(8) = 2
f(9) = 3

Amortized cost:
$\sum_{n=2}^{n} f(k) \leq \sum_{n=2}^{n} k$

Divide the equation with n

$$\leq log_2(\frac{n(n+1)}{2n})$$
$$\leq log_2(\frac{(n+1)}{2})$$

2 Question2

Answer

While pushing x, suppose we have k items stacked on the list and x is the largest which is the worst case then we have to check and remove all items,

which will take O(k) time. But each item can only be removed once, we can charge the item x 3 operations for every pushes using charging scheme. The 2 operations are when we check and find larger item then x at the top of the stack. As we know each time we push we could only find one item larger then x, and we only add x one time at the end. For the remaining operation it is used when the item is being compared to x in the stack and x is removed. When they are being compared we know that there only available 2 cases, either the item is smaller and is part of the initial 2 charged or the item is larger which will be applied with last operation charged to remove it from the list.
We know the total time for n pushes can be at most 3n, which is O(n). Therefore amortized cost of a push is O(1)

We could use the charging scheme with pop as well. When we pop off the top item from the stack we can consider it as using the last operation charged to the item. So any order of n pushes and pops will take at most O(n) or O(1) amortized time.

3 Question3

Answer
To check if undirected graph G = (V,E) is a tree or not we have to look for 2 things:
Does the graph have cycle or have more than one parent node.
Does all the nodes connected.

```
To check whether the graph is tree or not
    Data: n # The number of nodes
    visited <- false;
    if undirectedDFS(1, -1, visited) = false then
       return false;
    end
    for u <- 1 to n do
       if visited[u] = false then
          return false;
       end
    end
    return true;

To check whether the nodes have only one parent
    Data: u # The node to start from
          parent # The parent of u
          visited # Check if visited
    if visited[u] = true then
       return false;
    end
```

```
visited[u] <- true;
for child  neighbors(u) do
   if child != parent then
      result <- undirectedDFS(child, u, visited);
      if result = false then
         return false;
      end
    end
end
return true;
```

The running of above algorithm is O(V + E)

4 Question4

Answer

Let D be a DAG

Theorem

Let G = G(D), and D contains a directed path of length X(G) -1

Proof

Let D = (V,A) and A' $\subseteq$ A be a minimal set of edges such that D' = D - A is a DAG.
Assume that k is the longest directed path in D'. Represent c(v) is a length of longest path from v in D'. $c(v) \in$ (0,1,2,...,k).
We claim that c(v) is a proper coloring of G, which proved our theorem.

if D' contains path P = $(x_1, x_2, ..., x_k)$ then

$$c(x_1 \geq c(x_k) + k - 1 \tag{1}$$

Adding longest path Q from x_k to P to create path (P,Q) also uses the fact that D' is a DAG.

Suppose c is not a proper coloring of G and there is an edge vw $\in$ G with c(v) = c(w).

$\rightarrow$ vw $\notin$ A'. (1) implies c(v) $\geq$ c(w) + 1 - contradiction.

5 Question5

Answer

```
def DFS(G, s, t):
   visit[s] = true
   if s == t
      visit[s] = false, return 1
   cnt = 0
   for v is adj(s)
      if visit[v] == false
         cnt = cnt + DFS(G,s,t)
   visit[s] = false
   return cnt
```

6 Question6

Answer

The claim is true and can be prove using contradiction.

The statement mentioned that every node has at least n/2 degree, so we can assume the graph is not fully connected, meaning there will be some random node being separated from the main network.

Let n/2+1 forms a component which has been separated from others
$\rightarrow$n-(n/2+1) = n/2-1

As you can see it is impossible to form component B by themselves as there are n/2-1 nodes in it and each of the node in B should have one edge connect to A.This mean every node in B is being connected to A.

Therefore, we could say that all A and B are connected. Consequently the claim that G is connected is True.