Intro To Algo HW #1 - Saaif Ahmed

Thursday, January 23, 2020 8:43 PM

1: What is T(n) Exactly?

We show a few examples. By running this code in Python 3

```
def T(n):
    print("*")
    if n = = 0: return
    for i in range(n):
        T(i)
    return

T(n)
```

We can check the output of this code and find that the output, for example, for 2 is equal to ****. We can also input 3 and get an output of ********, or 8 '*' characters.

This is suggesting that the number of characters printed by $T(n) = 2^n$

Proof by Induction:

Base case: $P(0) = 2^0 = 1 = *$

Base Case is True

$P(n) \rightarrow P(n+1)$

Direct Proof :

Assume $P(n)$ is True

$P(n+1) = 2^{n+1} = 2 * 2^n = 2(2^n)$

$P(n+1)$ is True

So $T(n) = 2^n$

Intro to Algo HW#1

Thursday, January 23, 2020 8:49 PM

2: What is T(n) exactly?

We show a few examples of this algorithm by running this code in Python 3:

```
def T(n):
    if n = = 0: return
    for i in range(n):
        print("*")
    T(n-1)
    return

T(n)
```

We can check the output of this code and find that the output, for example, for 2 is equal to ***.

We can also input 4 and get an output of **********, or 10 '*' characters.

This is suggesting that $T(n) = sum\ of\ all\ numbers\ from\ 1 \rightarrow n$, or more formally written as: $T(n) = \frac{1}{2}(n)(n+1)$

Proof by Induction:

Base Case: $P(0) = \epsilon$ (empty string)

$$P(1) = \frac{1}{2}(1)(1+1) = 2\left(\frac{1}{2}\right) = 1 = *$$

Base Case is True:

$P(n) \rightarrow P(n+1)$

Direct Proof:

Assume $P(n)$ is True

$$\frac{1}{2}(n+1)\big((n+1)+1\big) = \frac{1}{2}(n+1)(n+2)$$

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

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

$P(n+1\ )$ is True

Thus $T(n) = \frac{1}{2}n(n+1)$.

Intro to Algo HW#1

Thursday, January 23, 2020 9:00 PM

3: Problem 3.6

A:

We show a proof that $\Sigma_{u \in V} d(u) = 2|E|$ for an undirected graph

Proof by Induction:

Base Case : $P(1)$ or just a root node.

There are no edges so $2|E| = 2(0) = 0 = \Sigma_{u \in V}$

Base Case is True

$P(n) \to P(n+1)$

Direct Proof:

Assume $P(n)$ is True

We analyze what happens when we connect a node.

From the Assumption we have a graph where $\Sigma_{u \in V} d(u) = 2|E|$
$\Sigma_{u \in V} d(u) = k \in \boldsymbol{N}$
$|E| = p \in \boldsymbol{N}$

So currently $k = 2p$. We now connect the next node. This adds 1 to p and 2 to the sum of all degrees. This is because a node connection adds 1 to the connecting node and the beginning node.

$k + 2 = 2(p + 1)$
$k + 2 = 2p + 2$

We know that $2p = k$ so by adding a new node to the graph we are essentially adding 2 to both sides of $k = 2p$. This preserves the $\Sigma_{u \in V} d(u) = 2|E|$ relationship.

B:

We know that $\Sigma_{u \in V} d(u) = 2|E|$ is a True relationship for all graphs of size $n \in \boldsymbol{N}$. We show that there must always be an even number of nodes with odd degree.

Proof by Contradiction:

We assume the opposite, that there must always be an odd number of nodes with odd degree.

The sum of all degrees from an odd numbers of nodes with an odd degree will always be an odd number. This cannot be true however due to the Handshaking Theorem where $\Sigma\ of\ degrees = 2|E|$. Because it is $2|E|$ the resultant number must be even.

Thus there cannot be an odd number of nodes with an odd degree.

Therefore the original statement is True.

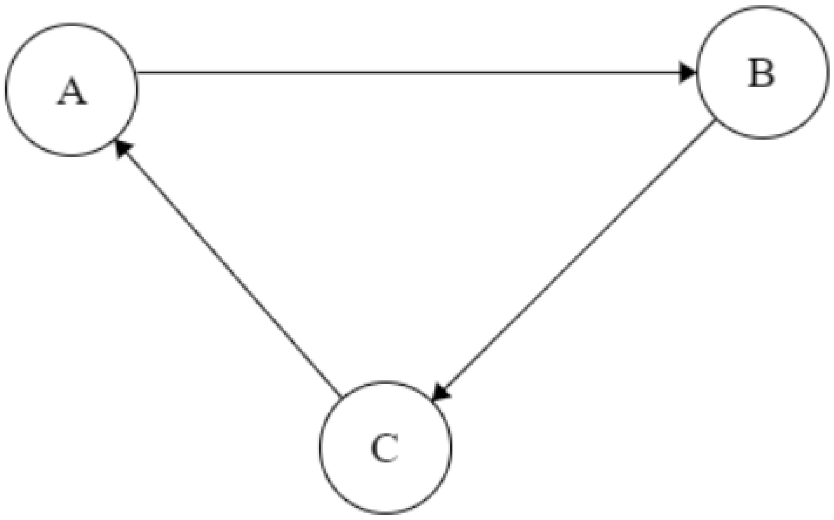
Intro to Algo HW#1

Thursday, January 23, 2020 9:42 PM

3: Problem 3.6 continued:

C:

Disprove we show a counter example



This shows an odd number of nodes with an odd in-degree. Thus the statement is false.

Intro to Algo HW#1

Thursday, January 23, 2020 9:44 PM

4:

We show pseudo code for an algorithm that finds the shortest distance between s & t

```
Algorithm #1:
    let length = 0
    let g1 = V1
    let g2 = V2
    let s = s
    let t = t

    #this labels the respective nodes in G1 and G2 that they have a connection
    for edge in $E'$:
        connect(edge, g1, g2)

    #use BFS
    let q = queue()
    length = 0
    q.append(s):
    while len(q) > 0:
        node = q.top()
        for edges in node:
            if edges->node.d = $\infty$:
                edges->node.d = node.d +1
                q.append(edges->node.d)

            if edges->node == t:
                length = node.d + 1
                break
```

Intro to Algo HW #1

Thursday, January 23, 2020 10:05 PM

5: Problem 3.7

A:

```
#we use a modified BFS

let g1 = V1
let g2 = V2

let bipartite = False
let x = queue()

let g = start_node(V1)

x.append(g)

while len(x) >0
    node = x.top()

    for edges in node:
        if edges->node.d = ∞:
            edges->node.d = node.d +1
            q.append(edges->node.d)

        if bipartite == True && (g1.in(edges->node)):
            return False
        elif biparite == False && (g2.in(edges ->node)):
            return False

        bipartite = !bipartite
```