COL351: Analysis and Design of Algorithms (CSE, IITD, Semester-I-2016-17) Homework-01

Name: Shivank Goel

Entry number: 2014CS10565

There are 6 questions for a total of 75 points.

---

1. Consider functions $f(n) = 10n2^n + 3^n$ and $g(n) = n3^n$. Answer the following:

(a) (½ point) State true or false: $f(n)$ is $O(g(n))$.

(a) True

(b) (½ point) State true or false: $f(n)$ is $\Omega(g(n))$.

(b) False.

(c) (2 points) Give reason for your answer to part (b).

(a) $10n2^n + 3^n \leq c\, n3^n \quad \forall n \geq n_0$ we have to find such c & n_0

We know that $10n2^n \leq 10n3^n \quad \forall n \geq 1$

$3^n \leq n3^n \quad \forall n \geq 1$

Adding we get $10n2^n + 3^n \leq 11n3^n \quad \forall n \geq 1$

Hence $f(n)$ is $O(g(n))$ where $c = 11$ & $n_0 = 1$

(b) $f(n) \geq c\, g(n) \quad \forall n \geq n_0$ We have to show no such c & n_0 exist

$10n2^n + 3^n \geq c\, n3^n \quad \forall n \geq n_0$

iff $10\left(\frac{2}{3}\right)^n + \frac{1}{n} \geq c \quad \forall n \geq n_0$

Claim however small c you take there will exist an n_1 s.t $10\left(\frac{2}{3}\right)^{n_1} + \frac{1}{n_1} < c$. This is because $\lim_{n\to\infty}\left[10\left(\frac{2}{3}\right)^n + \frac{1}{n}\right] = 0$ i.e for large n's this qty is going to dec & will be almost 0.

2. (2 points) A rooted tree has a special node r called the root and zero or more rooted subtrees $T_1, \ldots, T_k$ each of whose roots are connected to r. The root of each subtree is called the *child of* r and r is called the *parent* of each child. Nodes with no children are called *leaf* nodes. A *binary tree* is a rooted tree in which each node has at most two children.

Show by induction that in any binary tree the number of nodes with two children is exactly one less than the number of leaves.

Let's do Induction on total no. of nodes in binary tree.

$n = 2$

Let l be no. of leaves. Let m be no. of nodes with 2 children.

Here $l = 1$, $m = 0$ } Hence base condition is true

Fix $n = 1$ We may have $l = 1$, $m = 0$ Hence, true

Let the statement is true for $n = k, k-1, \ldots, 1$

$n = k+1$

Consider an arbitrary binary tree with $(k+1)$ vertices. There will exist a leaf node in this binary tree.

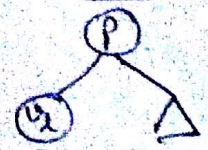
Proof: Suppose there is no leaf node. Every vertex has degree ≥ 2. Pick an arbitrary vertex. Move along. It's one of the edges. Now pick the another edge of this newly visited vertex. Continue this process. In this process no vertex can be repeated else there would be a cycle. The process will end somewhere because there are finite vertices. Thus we will end up to a leaf.

Case-I The parent p of that leaf vertex v_l has only 1 child.

- Now remove v_l. → we get tree with k vertices. satisfying $l_o - m_o = 1$ [o = old] [n = new]
- p is one of the leaves
- Now with v_l we have $l_n = l_o$ [p no more leaf, v_l is a leaf]
  $m_n = m_o$

$\therefore l_n - m_n = 1$

Case-II Parent p had 2 children. On removing v_l $l_o - m_o = 1$. On adding v_l $l_n = l_o + 1$ {v_l is new leaf} $m_n = m_o + 1$ {p is new deg 2 vertex} } Still $l_n - m_n = 1$ Hence Proved



8. (20 points) Given an undirected graph, we call a vertex critical if its removal disconnects the graph. Design an algorithm that finds all the critical vertices in a given graph. Give pseudocode, discuss, running time and give proof of correctness.

We will do a DFS of the given graph. It will form a DFS tree say T. We will note the discovery time of each vertex by $d(v), v \in V$. We will define a quantity called $low(v), v \in V$ where $low(v)$ will be equal to minimum discovery time of all the nodes that can be reached by a single edge (back edge or a tree edge) from the nodes in subtree rooted at v (including v). Now in this DFS tree T the root will be a critical vertex iff it has more than one children. Any other vertex in T will be a critical point iff some child of that vertex (say x is child of that vertex v) has $low(x)$ greater than or equal to $d(v)$.

In other words $low(v)$ is the min of discovery time of all the vertices that are either in subtree rooted at v or that are connected by any vertex in the subtree via a back edge. In practice $low(v)$ can be found as

$low(v) = \min\{ low(w) : w \text{ is a child of } v ; d(x) : v\text{-}x \text{ is a backedge} \}$

To Prove

$low(x) \geq d(v)$ is equivalent to saying subtree(x) has no backedge to any ancestor of v in T

$\Rightarrow$ $low(x) \geq d(v)$ tells that min discovery time of all vertices that in subtree(x) are connected by any vertex in subtree via back edge is greater than equal to discovery time of v. If some vertex had been connected to ancestor of v (say n) then low of that vertex had been lower than $d(v)$ because since n is ancestor $d(n) < d(v)$ & $(m\text{-}n)$ is a backedge $low(m) \leq d(n)$ & since m is in subtree(x) $\Rightarrow$ $low(x) \leq low(m) \leq d(n) < d(v)$ $\Rightarrow$ Contradiction

$\Leftarrow$ In subtree(x) $\forall x \in subtree(v)$ has a backedge. Now all descendants of v has $d(w) \geq v$. All vertices in subtree of v has $low(w) \geq v$ Hence proved.

Pseudocode

→ $d(v) = -1 \ \forall v$, $p(s) = -1$, DFS(s)

→ DFS(v)

```
d(v) = index++
low(v) = d(v)
For all edges (v,x)
    if (d(x) == -1)
        P(x) = v

        DFS(x)

        if (low(v) > low(x)) low(v)=low(x)
        if (low(x) >= d(v)) critical[v]=true
    else if (p(v) != x)
        if (low(v) > low(x)) low(v) = low(x)
```

→ For all edges (s, x)

```
    if (p(x) == s) c++
If (c>=2) critical[s] = true
```

Running time

- For initializing all the vertices' d(v) & critical(v) O(V)
- Each edge is processed 2 times (except edges from starting vertex which are processed 3 times) O(E)
- Overall Time Complexity O(E+V)

Proof of Correctness

(a) To prove root r is critical vertex iff r has ≥ 2 children in DFS tree

Proof $\Rightarrow$ Let r is critical vertex. To prove r has ≥ 2 children.
For sake of contradiction lets suppose r has atmost 1 child.

- If r has no child then there was only 1 vertex in G so r is not critical vertex.
- If r has 1 child say x. If we remove r only the tree edge (r, x) will be deleted & all the back edges to r will be deleted. All other tree edges will remain intact & vertices will be still connected, so r is not a critical vertex. Hence Contradiction

$\Leftarrow$ Let r has ≥ 2 children. Let u be the child of r whose discovery time $d(u)$ is earliest of all children. Let v be another child of r. We know that in a DFS all the vertices that are reachable from u will be discovered before before DFS for u ends. Since v is not a descendent of u in DFS tree thus the only path from u to v is via r. So r is an articulation or critical vertex.

(b) Any other vertex u is critical iff $\exists$ a child x of u s.t $low(x) >= u$
= Any other vertex u is critical iff $\exists$ a child x s.t subtree(x) has no back edge to some ancestor of u. (Equivalence proved earlier)

Proof $\rightarrow$ Let u is critical. Prove that $\exists$ a child x. Suppose no such child exist & every child has some back edge to some ancestor of u. Let $a_1, a_2, \ldots, a_n$ be children of u. Let p be parent of u. On removing u we get $(n+1)$ connected components of the T. But in graph G all $T(a_i)$'s are connected to $T(p)$ via a back edge & hence u is not a critical point. $\Rightarrow$ Contradiction

$\leftarrow$ $\exists$ a child x s.t subtree(x) has no back edge to any ancestor of u. For every vertex $w \in$ subtree(x) if (w, y) is an edge then y either $\in$ subtree(x) or $y = u$. Thus on removing $T(p)$ & $T(x)$ will be disconnected. u is critical

4. Consider two vertices s and t in a given graph. A pair of vertices (u, v) (different from s and t) are called bi-critical with respect to s and t if the removal of u and v from the graph disconnects s and t. Suppose we consider graphs where the shortest distance between s and t is strictly greater than $\lceil n/3 \rceil$. With respect to such graphs, answer the questions below:

(a) (5 points) Prove or disprove the following statement:

There exists a pair of vertices that are bi-critical with respect to s and t.

- True

For proving this lets assume $(P_1, P_2 \longrightarrow P_m)$ are the paths from s to t with distance of each path $> \lceil \frac{n}{3} \rceil$.

Consider all vertices at a particular distance k in all the paths where $k \leq \lceil \frac{n}{3} \rceil$. Let along the paths vertices at distance k are $\{a_{1k}, a_{2k}, \longrightarrow, a_{mk}\} = A_k$

→ We claim that for all such k's there must be atleast 3 distinct vertices in set A_k for s & t to be not bi-critical.

Proof → Else we will remove those 2 or 1 vertex at distance k & there will be no path left from s to t.

→ But now for each k we want atleast 3 distinct vertices where $k \in \{1, 2, \longrightarrow, \lceil \frac{n}{3} \rceil\}$

∴ No. of distinct vertices $\geq 1 + 3\lceil \frac{n}{3} \rceil + 1$

$$\geq \begin{cases} n+2 & n = 3m \\ 3m+3+2 = n+4 & n = 3m+1 \\ 3m+3m2 = n+3 & n = 3m+2 \end{cases}$$

$> n$

which is not possible & hence s & t will be bi-critical.

(b) (5 points) Design an algorithm for finding a bi-critical pair of vertices with respect to given vertices s and t. Discuss running time. Proof of correctness is not required.

Pseudocode.

→ $A = \{s\}$, $B = V \setminus \{s\}$, $X = \{s\}$

→ While ($B \mathrel{!}= \{\}$)

```
number = 0
temp = {}
for(all edges (x,y) such that x ∈ X and y ∈ B)
    temp = temp ∪ {y}
    number ++
    B = B \ y
If(number ≥ 3)
    A = A ∪ temp.
    X = temp
Else
    Print (bicritical vertices found)
    return temp
```

Running Time

Since all edges are processed once & also each node is visited. Running time is $O(V+E)$.

5. (20 points) A directed graph $G = (V, E)$ is called *one-way-connected* if for all pair of vertices u and v there is a path from vertex u to v or there is a path from vertex v to u. Note that the or in the previous statement is a logical OR and not XOR. Design an algorithm to check if a given graph is one-way-connected. Give pseudocode, discuss running time and give proof of correctness.

The idea is to find all the connected components in G & form the DAG G^{scc}. Now do the topological sorting of G^{scc}. In this topological sorting iff there exists an edge between every consecutive vertices then only graph is one-way connected.

Pseudocode for connected components.

→ First do a DFS marking the start and finish time of vertices.

```
DFS(u)
    start(u) = time++
    For all unvisited neighbours of u : (u,v)
        DFS(v)
    end(u) = time++
```

→ Reverse all the edges in G

→ Pick the maximum ending time vertex from G. Run a DFS from that vertex. All vertices that are marked visited in this process form one strongly connected component. Remove all this vertices from G & repeat the process till no more vertices are in G.

Pseudocode for topological sorting

→ Let v_1, v_2 ——— , v_k are vertices of G^{scc}

→ Since G^{scc} is a DAG, there will exist a vertex with indegree 0. Pick that vertex to get first element in topological sorting

→ Continue till no more vertices left

Pseudocode for one way connected

→ Find G^{scc}

→ Topologicall sort vertices of G^{scc}

→ If there exists an edge between every consecutive vertices

Print Yes else print NO.

Running Time

→ Strongly Connected Components $O(V+E)$ $O(k+e) \leq$

→ Let G^{scc} has k vertices $V_1, V_2 \rightarrow V_k$ and e edges $O(V+E)$

→ Checking edge between consecutive vertices $O(k)$

Overall time Complexity $O(V+E)$

Proof of Correctness

Basically we have to prove that a DAG is one-way connected iff & only if topological order has edges between all consecutive vertices.

⇒ DAG is one-way connected ie $\forall u,v \in G$ $\exists$ path from (u to v) or (v to u). Let topological sorting be $a_1, a_2 \rightarrow a_k$ Let $a_{i-1} \rightarrow a_i$ has no edge which is possible $a_i - a_{i-1}$ has a edge because it's a one-way connected, but that is not possible in topological sorting. Hence Contradiction.

$\Leftarrow$ Let topological order has edges between all consecutive vertices. To Prove its strongly connected.

Let $a_1, a_2, \ldots a_k$ be topological ordering

Now pick any two vertices a_i & a_j from this sequence. Let $(i<j)$ with loss of generality. Now we know that there is path from a_i to a_j because there is an edge between consecutive vertices. The path is given by $a_i \to a_{i+1} \to a_{i+2} \longrightarrow a_{j-1} \to a_j$

6. (20 points) Given a directed acyclic graph $G = (V, E)$ and a vertex u, design an algorithm that outputs all vertices $S \subseteq V$ such that for all $v \in S$, there is an even-length simple path from u to v in G. *(A simple path is a path will all distinct vertices.)*
Give pseudocode, discuss running time, and give proof of correctness.

Pseudocode

→ For each vertex $u \in G$ create two vertices u' and u'' in G'.

→ For each edge $(u-v) \in G$ add edges $(u'-v'')$ and $(u''-v')$ in G'

→ Start DFS from $s' \in G'$.

→ All the vertices marked as visited in G' as well as marked with single $'$, can be reached via even path length in original graph G.

Running Time

Since we are running DFS on G' running time is $O(2E+2V) = O(E+V)$.

Proof of correctness :

To Pr $u - v$ has even path length in G iff v' is marked visited in G'.

$\Rightarrow$ Let $s - v$ has even length path in G given by $s \to a_1 \to a_2 \dots \to a_{2m+1} \to v$ then there exist a corresponding path from s' to v' in G' given by $s' \to a_1'' \to a_2' \dots \to a_{2m+1}'' \to v'$ Hence v' will be marked visited in DFS (s') because it's reachable from s'.

$\Leftarrow$ Let v' is marked visited in G'. Then there exist a path from s' to v' given by $s' \to a_1 \to a_2 \dots \to a_n \to v'$. Since there are only 2 types of edges in G' is ($u' - v''$ or $u'' - v'$) the above path will be of form $\circledast$ $s' \to a_1'' \to a_2' \dots \to a_n'' \to v'$ There exist a corresponding path from s to v in G. $s \to a_1 \to a_2 \dots \to a_n \to v$. Let n was even but in eqn marked with $\circledast$ all even vertices are marked with single' but a_n is marked with'' Hence n must be odd. Thus in G s to v is a even path length sequence of vertices. Hence proved.