CS 580 Algorithm Design, Analysis, And Implementation
Fall 2021 Vassilis Zikas HW 1

Problem 1

Collaborators: List students that you have discussed problem 1 with: Vishnu Teja Narapareddy

(a) (i) $\log n, \ln n, \log 4n^3$

(ii) $2n^2 + n$

(iii) n^4

(iv) 3^n

(v) n!

(vi) n^n

Ranking of the function w.r.t n:
(i) < (ii) < (iii) < (iv) < (v) < (vi)

(b) f(n) and g(n) are always greater than 1 for every n $\geq 2^{200}$ (given)

For the upper bound:
$f(n) \leq \max(f(n), g(n))$ and g(n) $\leq \max(f(n), g(n))$
$\implies f(n) + g(n) \leq 2\max(f(n), g(n))$
$\implies f(n) + g(n) = O(\max(f(n), g(n)))$

For the lower bound:
For every n $\geq 2^{200}$, we can say that either f(n) would be greater than g(n) or vice versa.
$\implies f(n) + g(n) > \max(f(n), g(n))$
$\implies f(n) + g(n) = \Omega(\max(f(n), g(n)))$

From these two, we can say that:
$\implies f(n) + g(n) = \Theta(\max(f(n), g(n)))$

(c)

$$H_n = 1 + \frac{1}{2} + ... + \frac{1}{n}$$

For calculating the upper bound, assume k $= \lceil \log(n+1) \rceil$

$$\begin{aligned}
\implies H_n &\le 1 + \frac{1}{2} + ... + \frac{1}{2^k - 1} \\
&\le 1 + (\frac{1}{2} + \frac{1}{3}) + (\frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7}) + ... + (\frac{1}{2^{k-1}} + ... + \frac{1}{2^k - 1}) \\
&\le 1 + (\frac{1}{2} + \frac{1}{2}) + (\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4}) + ... + (\frac{1}{2^{k-1}} + ... + \frac{1}{2^{k-1}}) \\
&\le 1 + 1 + 1 + 1 + 1 + ... + 1 = k \\
\implies H_n &\le \lceil \log(n+1) \rceil \sim \log n \\
\implies H_n &= O(\log n) \qquad (1)
\end{aligned}$$

For calculating the lower bound, assume k $= \lfloor \log n \rfloor$

$$\begin{aligned}
\implies H_n &\ge 1 + \frac{1}{2} + ... + \frac{1}{2^k} \\
&\ge 1 + \frac{1}{2} + (\frac{1}{3} + \frac{1}{4}) + (\frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8}) + ... + (\frac{1}{2^{k-1}+1} + ... + \frac{1}{2^k}) \\
&\ge 1 + \frac{1}{2} + (\frac{1}{4} + \frac{1}{4}) + (\frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8}) + ... + (\frac{1}{2^k} + ... + \frac{1}{2^k}) \\
&\ge 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{2} + ... + \frac{1}{2} = 1 + + \frac{k}{2} \\
\implies H_n &\ge 1 + \frac{1}{2} \lfloor \log n \rfloor \sim \frac{1}{2} \log n \\
\implies H_n &= \Omega(\log n) \qquad (2)
\end{aligned}$$

From (1) and (2), we can conclude that $H_n = \Theta(\log n)$

Problem 2

Collaborators: List students that you have discussed problem 2 with: Vishnu Teja Narapareddy

(a) To find the longest path in the country, I find the longest path from each of the cities and then return that path which was longest among all of them.
Treat the cities as vertices of graph and the railways lines connecting them as edges between them
In my approach, I run the function BFS for every node in the graph and keep on updating the distance and path returned from that function if it is greater than the current one.
Function BFS consists of 3 arrays $\rightarrow$ visited, distance and parent, and a queue. The given vertex is added in the queue and while loop runs until all it's neighbours are marked visited and their respective distance and parent are updated. After that, the function finds for the most distant vertex from the given node using the distance array and generate the path by using the parent array.

Analysing the time complexity:
BFS function goes through over all the nodes in the graph once. So this means the function takes O(n) time where n is the number of nodes in the graph.
LongestPath function runs the BFS function for every node. So this means that overall the algorithm takes O(n^2) time.

Algorithm 1: Find the longest path in the country

Input: Graph G
Output: Longest path in G along with distance

```
Function LongestPath(G):
    currPath ← null
    currDist = 0
    for each vertex v in graph G do
        path, dist ← BFS(G,v)
        if dist > currDist then
            currDist = dist
            currPath ← path
    return currPath, currDist

Function BFS(G, s):
    take array visited of size V and initialise to false
    take array distance of size V and initialise to ∞
    take array parent of size V and initialise to null
    let path be a resizeable array
    let Q be a queue
    add s to Q and mark s as visited
    while Q is not empty do
        remove vertex v from Q
        for all neighbours n of v in G do
            if n is not visited then
                add n to Q
                mark n as visited
                distance[n] = distance[v] + 1
                parent[n] = v
    maxDist = 0
    maxVertex = null
    for each vertex v in distance array do
        if distance[v] ≥ maxDist then
            maxDist = distance[v]
            maxVertex = v
    while parent[maxVertex] is not null do
        add parent[maxvertex] to path
        update maxVertex to parent[maxVertex]
    reverse the path array
    return path, maxDist
```

(b) To prove : the longest railway path starting from x is either from x to a or from x to b, where a and b are ends of the longest railway path in the country
Proof :
There are two cases : either x lies on path between a and b or it does not

(a) x lies on the path between a and b
Let say the longest path from x does not end at either a or b. Suppose, it ends at another node c.
This implies that we can extend the path from x to a to x to c which means that path between b and c is longer than b and a (Contradiction)
$\implies$ Longest path from x ends either at a or b

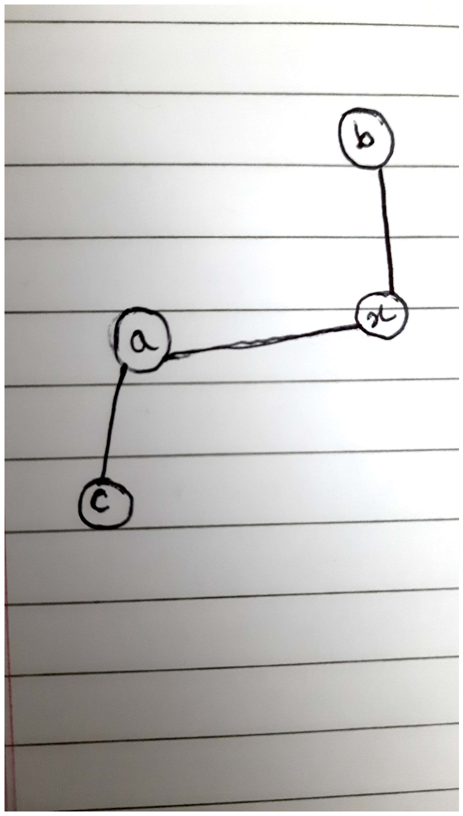

Figure 1: Case (a)

(b) x does not lie on the path between a and b

i. longest path from x touches path from a to b
Let say the node at which longest path from x touches the path from a to b is m.
We can say that:
longest path from x = path from x to m + longest path from m
From case (a), we already know that longest path from a node in between the diameter of the tree ends at one of the diameter.
$\implies$ Longest path from x in this scenario also ends at either a or b.

ii. longest path from x does not touch path from a to b
Let say the longest path from node x ends at y. We have two non-intersecting paths in the tree.
Now take two nodes m and n, such that these nodes connect these two paths. Node m is on path x

to y and is closer to node x whereas node n is on path a to b and is closer to b
We can say that:
longest path from n ends at a   (using case (a))
longest path from m ends at y   (otherwise x won't have y as its farthest node)
Since path length from m to y (p_1) > path length from m to a ($p_2 + p_3$), and upon extending the path y to m up to n ($p_1 + p_3$), we get that $p_1 + p_3 > p_2$
This means that longest path from n ends at y.   (Contradiction, already proved in case (a))
$\implies$ Longest path from x in this scenario also ends at either a or b.

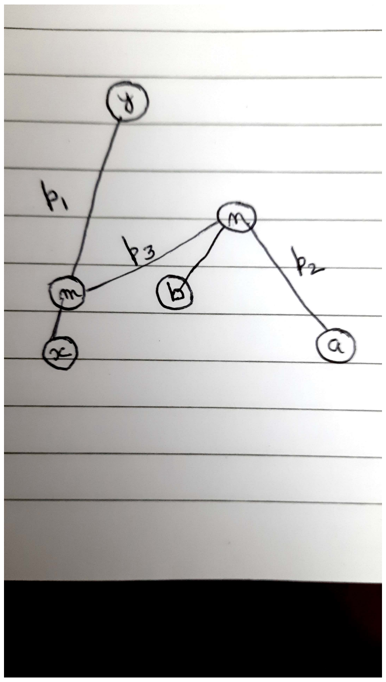

Figure 2: Case (b) (ii)

Hence, proved

(c) For this algorithm, I have used the result derived in the earlier part which says that longest path from any vertex in graph ends at one of ends of the longest path in the overall graph.
So the functionality of the BFS function remains same as described in 2(a).
In the LongestPath function, I call the BFS function for any random node in the graph and the last vertex in that path would be one of the ends of the longest path in the overall graph. After this, BFS function is called again for that last vertex and this call would give us the longest path in the overall graph.

Analysing the Time complexity:
BFS function goes through over all the nodes in the graph once. So this means the function takes O(n) time where n is the number of nodes in the graph.
LongestPath function calls the BFS function twice in its implementation. So this means that overall the algorithm takes O(n) time.

Algorithm 2: Find the longest path in the country

Input: Graph G

Output: Longest path in G along with distance

```
Function LongestPath(G):
    path ← null
    dist = 0
    take any vertex v in G:
        path, dist ← BFS(G,v)
    take the last vertex in path, let say t:
        path, dist ← BFS(G,t)
    return path, dist

Function BFS(G, s):
    take array visited of size V and initialise to false
    take array distance of size V and initialise to ∞
    take array parent of size V and initialise to null
    let path be a resizeable array
    let Q be a queue
    add s to Q and mark s as visited
    while Q is not empty do
        remove vertex v from Q
        for all neighbours n of v in G do
            if n is not visited then
                add n to Q
                mark n as visited
                distance[n] = distance[v] + 1
                parent[n] = v
    maxDist = 0
    maxVertex = null
    for each vertex v in distance array do
        if distance[v] ≥ maxDist then
            maxDist = distance[v]
            maxVertex = v
    while parent[maxVertex] is not null do
        add parent[maxvertex] to path
        update maxVertex to parent[maxVertex]
    reverse the path array
    return path, maxDist
```

Problem 3

Collaborators: List students that you have discussed problem 3 with: Vishnu Teja Narapareddy

(a) No, Lil Omega should not accept the proof provided by Lil Delta.
The given proof makes incorrect use of mathematical induction to prove the statement that all students have same eye colour.
The flaw in the proof is that it assumes that the two subsets of n students out of n+1 students would share a common element in between them always. However, this is not true if we take n+1 = 2.
In that case, we can have one student with black eye colour and other with brown eye colour. So, both of them would satisfy the base condition of P(1) but they won't satisfy P(2) as now there would be no common element in between them.
Hence, the proof given here is not acceptable.

(b) P(n) : In a full m-ary tree with n nodes, L(T) = (m-1)I(T) + n-1
Now, let us prove this by mathematical induction:
P(1) : only root node is there in the tree, n=1
L(T) = 0 as well as I(T) = 0
So, this is true for P(1).

Now, let's assume that the given equation is true for P(n), i.e
L(T) = (m-1)I(T) + n-1

Now, we will prove for P(n+m) because in a full m-ary tree, each node has either m children or 0 children. So the next stage in the graph would have n+m vertices.
Let say, we added m children to a leaf node which was at depth d in the tree.

$$I^{'}(T) = I(T) + d$$

that leaf node now becomes internal node

$$L^{'}(T) = L(T) - d + m(d+1)$$

that leaf node now becomes internal node
m new leaf nodes are added at depth d+1

$$L^{'}(T) = (m-1)I(T) + n - 1 - d + m(d+1)$$

$$L^{'}(T) = m * I(T) - I(T) + n - 1 - d + md + m$$

$$L^{'}(T) = m * (I(T) + d) - (I(T) + d) + n + m - 1$$

$$L^{'}(T) = (m-1) * I^{'}(T) + (n + m - 1)$$

$\Longrightarrow$ P(n+m) is also true

Hence, proved.

Problem 4

Collaborators: List students that you have discussed problem 4 with:None
Yes