CS345 Assignment 3

Siddharth Agrawal (150716)
AviRaj (150168)

September 24,2017

Question 1

Part 2

Algorithm

Algorithm 1 *Algorithm to assign weights to the edges in $G = (V, E)$ such that all edges from s to t have unique pathids*

function UNIQUEPATHID(G, s, t)
    Apply topological ordering algorithm on graph G ▷ As given in class
    $Num[t] \leftarrow 1$
    for each $v \in V \backslash \{t\}$ in decreasing order of the value of τ_v **do** ▷ τ_v = topological number of vertex v
        $Num[v] \leftarrow 0$
        for each edge $(v, w) \in E$, $w \in V$ **do**
            $w[v, w] \leftarrow Num[v]$
            $Num[v] \leftarrow Num[v] + Num[w]$
        end for
    end for
    return w
end function

In the above algorithm, the meaning of the various variables used are -

G = The given DAG
V = Set of vertices in G
E = Set of edges in G
s = Root vertex of the DAG G
t = Exit vertex of the DAG G
Num = array storing the number of paths to the exit vertex t, i.e., $Num[v]$ = number of distinct paths from v to t
w = 2D array storing the egde weights, i.e., $w[v, w]$ = weight of edge (v, w)

Proof of Correctness

Lemma (1.1). *$Num[v]$ in the above Algorithm denotes the number of distinct paths from v to t.*

Proof. We prove this lemma using induction on topological number τ in decreasing order.
Base Case.
$\tau_v = |V| - 1$. Here $v = t$, i.e., the exit vertex. Hence $Num[v] = 1$ is trivial.
Induction Step.
Induction Hypothesis: $Num[w]$ follows the given lemma $\forall w \in V$ such that $i \leq \tau_w \leq |V| - 1$
let $v \in V$ and $\tau_v = i - 1$.
Number of paths from v to $t = \sum_{(v,w)\in E}$ Number of paths from w to t.
Using induction hypothesis -
Number of paths from v to $t = \sum_{(v,w)\in E} Num[w] = Num[v]$(As per the above Algorithm).
Hence $Num[v]$ denotes the number of distict paths from v to t.
$\square$

Theorem (1.2)**.** *In the above Algorithm, when we assign weights to the edges originating from v, then it is ensured that the pathids of all the paths from v to t are **unique** and have value in the range* 0 *to* $Num[v] - 1$.

Proof. We prove this theorem using induction on topological number τ in decreasing order.
Base Case.
$\tau_v = |V| - 1$. Here $v = t$, i.e., the exit vertex. Hence theorem is trivially satisfied since only one path is present.
Induction Step.
Induction Hypothesis: The theorem holds $\forall w \in V$ such that $i \leq \tau_w \leq |V| - 1$.
Consider $v \in V$ such that $\tau_v = i - 1$.
Suppose edges from v end up at vertices $w_1, w_2, ..., w_k$.
Consider edge (v, w_j), $\quad j \in [k]$
Using induction hypothesis, pathids of all paths from w_j to t uniquely lie between 0 to $Num[w_j] - 1$.
As per our Algorithm, weight of edge $(v, w_j) = \sum_{l=1}^{j-1} Num[w_l]$.
Therefore, all paths from v to t, starting with edge (v, w_j), have unique pathids in the range $\sum_{l=1}^{j-1} Num[w_l]$ to $\sum_{l=1}^{j} Num[w_l] - 1$.
Hence pathids of any path from v to t lie uniquely in the range 0 to $Num[v] - 1$ ($\because Num[v] = \sum_{l=1}^{k} Num[w_l]$).
$\square$

Using the above theorem, we get that all paths from s to t have unique pathids in the range 0 to $N - 1$. Thus our algorithm is correct.

Complexity Analysis

Time Analysis:
Topological Ordering takes $O(m + n)$ time.
In the rest of the Algorithm, since we run a loop on the number of vertices ($O(n)$) and in each loop we inspect all edges ($\sum O(m_i) = O(m)$), hence its time complexity is $O(m + n)$.
Hence total time complexity of our Algorithm is $O(m + n)$.

Space Analysis:
We only need $O(n)$ extra space to maintain the array Num which stores the number of paths to exit vertex t.

Question 2

Part 2

Algorithm

Algorithm 2 Algorithm to check if the graph G is a unique path graph, given vertex s reachable to all other vertices

```
function DFS(v)
    Visited[v] ← true
    D[v] ← count++
    HighPoint_1[v] ← D[v]
    HighPoint_2[v] ← D[v]
    for each edge (v, w) do
        if Visited[w] = false then
            (HP_1, HP_2) ←DFS(w)
            if HP_1 < HighPoint_1[v] then
                if HP_2 < HighPoint_1[v] then
                    HighPoint_2[v] ← HP_2
                else if HP_2 < HighPoint_2[v] then
                    HighPoint_2[v] ← HighPoint_1[v]
                end if
                HighPoint_1[v] ← HP_1
            else if HP_1 < HighPoint_2[v] then
                HighPoint_2[v] ← HP_1
            end if
        else
            if Finished[w] = true then
                UniquePathGraph ← false
                break
            else
                if D[w] < HighPoint_1[v] then
                    HighPoint_2[v] ← HighPoint_1[v]
                    HighPoint_1[v] ← D[w]
                else if D[w] < HighPoint_2[v] then
                    HighPoint_2[v] ← D[w]
                end if
            end if
        end if
    end for
    if HighPoint_2[v] < D[v] then        ▷ ∃ atleast 2 backedges from the subtree rooted at v to its ancestors
        UniquePathGraph ← false
    end if
    Finished[v] ← true
    return (HighPoint_1[v], HighPoint_2[v])
end function
function Main()
    UniquePathGraph ← true
    for each v ∈ V do
        Visited[v] ← false
        Finished[v] ← false
    end for
    count ← 0
    DFS(s)
    if UniquePathGraph = true then
        print: G is a unique-path graph
    else
        print: G is not a unique-path graph
    end if
end function
```

In the above Algorithm, the meaning of the various variables used are -

$Visited$ = Array to check whether a vertex has been visited or not
$HighPoint_1$ = $HighPoint_1[v]$ stores the earliest ancestor to which there exists an edge from v
$HighPoint_2$ = $HighPoint_2[v]$ stores the second earliest ancestor to which there exists an edge from v
D = Array to store the order of appearance of a vertex by the DFS algorithm
$Finished$ = Array to determine whether the DFS algorithm has exited the subtree rooted at any vertex
$UniquePathGraph$ = Label to determine if the graph G is a unique-path-graph or not
s = The vertex reachable to all the other vertices in the given graph G

Proof of Correctness

Theorem (2.1)**.** *A vertex v has atmost one path to all other vertices iff-*

1. *v has NO forward edge.*
2. *v has NO cross-edge.*
3. *Subtree rooted at v has ATMOST ONE vertex with backedge to v's ancestors.*

Proof. Statement 1 and 2 have already been proved in class (if v has a forward/cross edge, then there exists $w \in V$ such that there are more than one paths from v to w).
Now we prove Statement 3 using 'proof by contradiction'.
Assume that Statement 3 is false, i.e., $\exists$ a vertex v such that it has two distinct vertices w_1 & w_2 in its rooted subtree, which have backedges to v's ancestors.
Note: w_1 (or w_2) can also be the rooted vertex v itself. We have just considered the most general case
Let the backedges be - (w_1, x_1) & (w_2, x_2) (Note: $D[x_1] < D[v]$ & $D[x_2] < D[v]$)
Let $D[x_1] < D[x_2]$, i.e., x_1 is the ancestor of x_2.

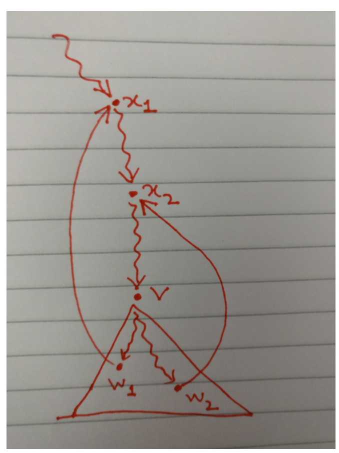

Figure 1: figure depicting the general case assumed

It can be clearly seen that $\exists$ 2 'distinct' paths from v to x_2 viz.-

1. $v \rightsquigarrow w_1 \rightarrow x_1 \rightsquigarrow x_2$
2. $v \rightsquigarrow w_2 \rightarrow x_2$

Hence our initial assumption was wrong. This means that Statement 3 holds true (by Contradiction).

□

Using Theorem (2.1), we can see that our algorithm is correct (since it checks whether or not there are two backedges from subtree(v) to v's ancestors).

Complexity Analysis

Time Analysis:
In the above DFS search performed, each vertex and edge is visited only once.
Hence time complexity of the algorithm = $O(m + n)$

Space Analysis:
In the above Algorithm, only $O(n)$ space arrays are used (namely- $Visited, HighPoint_1, HighPoint_2, D, Finished$).
Hence space complexity = $O(n)$.