CS345: Data Structures and Algorithms II 3
Indian Institute of Technology Kanpur
Some petty course title, 17–18

ASSIGNMENT

1

Student Name: Gurpreet Singh
Roll Number: 150259
Date: September 23, 2017

Question 1

Pseudocode

Algorithm 1 Assigning weights to edges for unique path ids

```
1  procedure UniquePathID(G, s, t)        ▷ where G is the DAG, s amd t are the root and exit vertex
2      T = TopologicalSorting(G)          ▷ As discussed in class
3
4                  ▷ Loop through the vertices in the topological sorting in the reverse order, i.e. sink to source
5      for v in reverse(T) do
6          if Outgoing[v] = 0 then        ▷ Outgoing holds the number of outgoing edges from the vertex v
7              if v = t then
8                  pathCount[v] = 1       ▷ pathCount holds the number of paths from the vertex v to t
9              else
10                 pathCount[v] = 0
11             end if
12         else
13             pathCount[v] = 0
14             for e = (v, w) ∈ E do
15                 e.weight = pathCount[v]          ▷ Assigning weight to edge e
16                 pathCount[v]+ = pathCount[w]
17             end for
18         end if
19     end for
20 end procedure
```

Proof of Correctness

Claim *At every iteration of the outer loop, pathCount[v] stores the number of paths from vertex v to t and assigns weights such that each path from v to t has unique pathid from 0 to pathCount[v] − 1*

Proof

We consider induction on the number of vertices in the graph. For any vertex, we will remove that vertex and treat all the subtrees formed.

Base Case: The number of vertices is 2, i.e the source vertex s and the sink vertex t. Since there is one and only one path in this case, the weight on the edge (s, t) will be zero, and path count from and our claim is true.
Induction Hypothesis: Suppose our claim holds for all graphs which have less than n vertices in the graph.
Induction:

Consider all vertices w_i such that $(v, w_i) \in E$. If we remove v from the graph and assume w_i to be the source vertex (one by one), our induction hypothesis holds. Hence the vertices w_i will hold all the paths count from w_i to t and weights are assigned to edges such that each path from w_i to t has a unique pathid from 0 to $pathCount[w_i] - 1$.

> Since the the number of paths from v to t will be nothing but the sum of the number of paths from w_i to t for all i, the definition of *pathCount* is well held. Now since for every edge (v, w_i), we add the number of paths of w_k for all $k \in [i]$, it is impossible to get any path that has its pathId equal to the pathId of any other path. Since the paths from w_k to t will have pathId $<$ the weight of the edge (v, w_i). Hence, we can say that all paths from v to t have distinct path ids. Also, the minimum pathId will be 0 and the maximum pathId will be $pathCount[v] - 1$. This can be easily observed from the weights of the edges from v.
>
> Hence, our claim holds for this graph also. Therefore, via induction, we proved that our claim is true for all DAG graphs with a source and a sink vertex.

As can be seen, at the end of the iteration for the source vertex s, $pathCount[s]$ will store the number of paths from s to t and weights would be assigned to edges such that each path from s to t will have unique pathid from 0 to $N - 1$ where N is number of paths from s to t.

Analysis

As discussed in class, computing topological ordering of the given DAG will have $O(m + n)$ time complexity. For any given vertex, we do $O(m_i)$ computation where m_i is the number of outgoing edges plus some $O(1)$ computations. Since each edge is handled only once, the total time complexity for the later part of the algorithm is $O(m + n)$. Therefore, the total time complexity of the given algorithm is $O(m + n)$.

Question 2

Pseudocode

Algorithm 2 Augmenting DFS to check if graph is a unique path graph

```
1   globalCount = 0
2   uniquePathGraph = TRUE
3
4   procedure DFS(u)
5       D[u] = globalCount++                    ▷ D stores the Start Time / DFN Number of a vertex
6       visited[u] = TRUE
7
8       ancestor[u] = −1
9
10      for each each (u, v) ∈ E(G) do
11          if visited[v] then
12              if finished[v] then
13                  uniquePathGraph = FALSE
14                  break
15              else
16                  if D[v] < D[u] then/        ▷ If v is an ancestor of u
17                      if ancestor[u] ≠ −1 then
18                          uniquePathGraph = FALSE
19                      else
20                          ancestor[u] = D[v]
21                      end if
22                  end if
23              end if
24          else
25              DFS(v)
26              if ancestor[v] < D[u] then
27                  if ancestor[u] ≠ −1 then
28                      uniquePathGraph = FALSE
29                  else
30                      ancestor[u] = ancestor[v]
31                  end if
32              end if
33          end if
34      end for
35
36      Finished[v] = TRUE
37  end procedure
38
39  procedure isUniquePathGraph(s)
40      uniquePathGraph = TRUE
41      DFS(s)
42
43      return uniquePathGraph
44  end procedure
```

Proof of Correctness

Consider a DFS tree of the graph, with the DFS starting at s, the source vertex. We do not need to worry about the tree edges as the DFS tree is a unique path graph. The only additions would be the non-tree edges. There can be three types of non-tree edges, namely "Forward Edge", "Cross Edge" and "Back Edge". Let us consider all three cases.

Case 1 *If there is at least one forward edge in G*

> Let this edge be (u, v). Since this is a forward edge, there must be a path from u to v in the DFS tree, i.e. u must be an ancestor of v. Consider this path to be a sequence of vertices $v_1, v_2 \cdots v_k$ where $v_1 = u$ and $v_k = v$. Since $(u, v) \in E(G)$, clearly this is another path from u to v.
>
> Hence in this case, we can say that the graph G cannot be a unique path graph.

Case 2 *If there is at least one cross edge in G*

> Let this edge be (u, v). Since this is a cross edge, the vertices u and v belong to different subtrees, and will have at least one common ancestor (i.e. LCA). Let $w = LCA(u, v)$. Therefore, there is a path from w to u say $v_1, v_2 \cdots v_k$ where $v_1 = w$ and $v_k = u$. There will also be a path from w to v which does not include u as they are in two different subtrees rooted at children of w. Consider this path, say $v'_1, v'_2 \cdots v'_{k'}$, where $v'_1 = w$ and $v'_{k'} = v$.
>
> Since $(u, v) \in E(G)$, we can find a path from w to v going through u, i.e. $v_1, v_2 \cdots v_k, v$, where $v_1 = w$ and $v_k = u$. Hence there are two paths from w to v.
>
> Hence, in this case too, we can say that the graph G cannot be a unique path graph,

Case 3 *If there is at least one back edge in G*

> Nothing can be inferred directly in this case, as already discussed in the class. However, there are cases in which multiple back edges will eliminate the chance of a unique path graph.

Our code is handling the first two cases in the part where we are checking if the vertex v is visited and finished as well. This is because any cross edge and forward edge to a vertex v assures that v has been visited and its DFS has finished.

Note: *From here on, we assume that the graph G has no forward or cross edge.*

The additional cases are when we have two back edges from a subtree. We are defining a back edge from a subtree as follows

Definition 2.1 *Consider a subtree of the DFS tree rooted at vertex u. An edge (v, w) is a back edge from this subtree if $v \in Tree(u)$ and $w \notin Tree(u)$, and w is an ancestor of u in the DFS tree.*

Claim *G is not a unique path graph if for any vertex $u \in V(G)$, there are at least two back edges from the subtree rooted at u in the DFS tree of G*

Proof

> Let us say that the back edges are $(v_1, w_1), (v_2, w_2) \in E(G)$. Here the two edges follow the definition provided above.
>
> Since both w_1 and w_2 are ancestors of u, it is not possible that they are parts of different subtrees in the DFS tree. Hence, either $w_1 = w_2$ or one is the ancestor of the other. Without loss of generality, we can assume that $w_2 \in Tree(w_1)$. This generalizes both the cases.
>
> Since $v_1, v_2 \in Tree(u)$, we can say that there is a path from u to both the vertices. Also, there are back edges from v_1 to w_1 and v_2 to w_2. Hence, we can say that there is a path from u to both w_1 and w_2 (not passing through w_2).
> Now, since $w_2 \in Tree(w_1)$, there is a path from w_1 to w_2 not passing through u (as w_2 is an

ancestor of u, and therefore there will be such a path in the DFS tree). Therefore, we have a path from u to w_2 passing through w_1 as well. Hence we have found two paths from u to w_2. Therefore, G cannot be a unique path graph in this case. Hence our claim is true.

Note: *We are representing the subtree rooted at u as $Tree(u)$*

Claim *G is a unique path graph if there is no vertex $u \in V(G)$ such that there are two back edges from the subtree rooted at u*

Proof

We will prove the contrapositive of this statement. Therefore, we assume that the graph G is not a unique path graph. Then, we need to prove that there is at least one vertex $u \in V(G)$ such that there are at least two back edges from the subtree rooted at u.

Since the graph is not a unique path graph, there is a pair of vertices, say u and v, such that there are two distinct paths from u to v.

If the vertices are located in two different subtrees in the DFS tree of the graph, then there must be at least one cross edge. This violates our assumptions. Also, if u is an ancestor of v, then there must be either a cross edge or a forward edge. This also violates our assumptions. Hence, we can say that v has to be the ancestor of u.

Since v is an ancestor of u, for any path from u to v, there has to at least one back edge in this path. Also, if the back edge is from a vertex, say w, then $w \in Tree(u)$, otherwise, there will be some other vertex for which this is true. Therefore, this back edge is a back edge from the subtree rooted at u.

This is valid for both the paths from u to v. Hence we have two back edges from the subtree rooted at u. These two back edges are distinct because the paths are distinct, and cannot have any common edge or vertex.

This proves the contrapositive statement, and hence our claim.

This is handled in our code in the part where we check for the reachable ancestor (ancestor of subtree of child node) for all the children of a node, and for all back edges of a node. If the reachable ancestor is an ancestor of this node (checked using the DFS number), we can say that this is a valid reachable ancestor for this node. If for any node, we have already set the ancestor vertex, and we get another reachable ancestor, we reach the first claim, and hence output false.

Using the above claims and conditions, we can conclude —

- The graph G is not a unique path graph if there is any forward edge in any DFS tree
- The graph G is not a unique path graph if there is any cross edge in any DFS tree
- The graph G is not a unique path graph if there is any vertex u such that there are two back edges from the subtree rooted at u in the DFS tree
- The graph G is a unique path graph otherwise

Since our code handles all the above conditions, we can say that the algorithm is correct, and outputs TRUE iff the graph G is a unique path graph, else returns FALSE.

Time Complexity Analysis

Since we are not using anything other that simple $O(1)$ augmentation in DFS, the time complexity will remain the same as in DFS which is $(|V| + |E|)$ as discussed in class.