CS345 : Theoretical Assignment 3

Pranjal Prasoon (150508)
Raktim Mitra (150562)

September 2017

1 Question 1.2 (Hard Version)

1.1 Algorithm

Algorithm 1: Assigning edge weights for pathid

$s \leftarrow$root vertex
$t \leftarrow$exit vertex
$T(v)$ = Array storing vertices in reverse order of topological sort
$Weight[n]$ = Array storing edge weights
$Path_{num}[v]$ = Array storing number of paths from v to t with $Path_{num}[t] = 1$
Function *assign weights(T)*
  We start accessing the array T in decreasing order of topological numbering
  for *vertex v in T* **do**
    if $v = t$ **then**
      $Path_{num}[v] \leftarrow 1$
  else
    $Path_{num}[v] \leftarrow 0$
    for *every edge $e = v \to w$* **do**
      $Weight[e] \leftarrow Path_{num}[v]$
      $Path_{num}[v] \leftarrow Path_{num}[v] + Path_{num}(w)$

1.2 Time complexity analysis

Topological Sort takes $T_1 = O(m + n)$ time and we traverse the array T once for every vertex and then, we acess every outgoing edge for every vertex, giving us a time complexity of $T_2 = O(m + n)$.

Therefore, overall time complexity, $T = O(m + n)$.

1.3 Proof of Correctness

Lemma 1 *For the exit vertex t, we just assign $Path_{num}$. For any other vertex v, when the algorithm is done processing v, we claim the following:*

- *$Path_{num}$ stores the number of paths form v to t and $Path_{num}[t] = 1$*
- *Sum of weights along each path from v to t generates a unique value $\in [0, Path_{num}[v]-1]$,which is the **pathid** of that path from v to t.*

Proof *We apply induction on the position of the vertex in topological sort. Let height of the vertex mean its distance from the end in the array T ($height(t) = 0$, obviously).*

- ***Base Case*** *$height(v) = 0$. For this case, $v = t$ and both the claims are trivially true.*
- ***Induction Step*** *Let this claim hold for all vertices of height $\leq h$.*

  Now, consider a vertex v with $height(v) = h + 1$ and successors $= w_1, w_2, ..., w_k$. Now, each of w_i has a height $\leq h$ (DAG) and so, it satisfies both our claims. Now, at the end of the iteration for v, $Path_{num}[v] = \sum_{i=1}^{k} Path_{num}[w_i]$. So, the first claim holds.

  *Also, for each w_j, sum of weights along each path from w_j to t generates a unique value $\in [0, Path_{num}[w_j] - 1]$ (since $height(w_j) \leq h$) . So, sum of weights along each path from v to t including $v \rightarrow w_j$ (assuming that when we say w_j, we mean that all vertices from w_1 to w_{j-1} have already been visited in the algorithm) is essentially in the range $[\sum_{i=1}^{j-1} Path_{num}[w_i], \sum_{i=1}^{j} Path_{num}[w_i] - 1]$. Since $Path_{num}[w_i] > 0$ for all is and edge weights are integral, so, these intervals are mutually disjoint and hence all paths from v to t will have a unique **pathid**. Also, sum of weights along each path from v to t will lie in $[0, \sum_{i=1}^{k} Path_{num}[w_i]-1]$ and as $Path_{num}[v] = \sum_{i=1}^{k} Path_{num}[w_i]$, it lies in $[0, Path_{num}[v]-1]$. Hence, v also satisfies both our claims.*

This sets up induction and proves our claims.

2 Question 2.2 (Hard Version)

2.1 Algorithm

- Perform DFS on the vertex **u**. All vertices of the graph will be included in its DFS tree (by definition of **u**).
- Since all vertices are present in the DFS tree of **u**, so, as we have seen in the class, the presence of even one **Forward** or **Cross** edge violates the unique graph property.
- For **Back** edges, we say that if from a vertex v_1 there is a back edge to its ancestor u_1 and from a vertex v_2 there is back edge to its ancestor u_2 and all of the 4 are in some sort of ancestor-descendant relationship with each other (all belong to same branch of the tree), both edges **overlap** if the **closed** intervals $[DFN[u_1], DFN[v_1]]$ and $[DFN[u_2], DFN[v_2]]$ have anything in common. And if we find such overlapping cases, then we claim that unique path property is violated.

  Now, this overlap means that **there will atleast be one vertex such that there are at least two back edges from the subtree rooted at that vertex to its ancestors**. This is easy to obsere because let's say that both the intervals have a particular vertex **v** in common, then that vertex v satisfies this condition, always.
- If we don't encounter any of the situations above, we report that the graph is a unique path graph.

Algorithm 2: Modifying DFS algo to find overlap

```
high[n] = Array storing highest ancestors (in the DFS tree) in the tree rooted at v
count[n] = Array storing count, initialized with xero
Parent[n] = Array stroing the parent of each vertex in the DFS tree
visited[n]= Our standard visited array for DFS, intiialized to zero
count = variabe for calculating DFN, initialized to zero.
DFN[n] = Array storing DFN numbers.
F[n] = Array storing finish times.
Finsihed[n] = Array initialized with false for each v.
Function DFS(v)
    visited[v] ← 1
    DFN[v] ← count++
    high[v] ← ∞
    for each neighbour w of v do
        if visited[w] and Parent(v) ≠ w then
            if Finished[w] then
                Output: "Not a unique path graph"
                break;
            high[v] = min(high[v], DFN[w])
            count[v]++
        else if !visited[w] then
            Parent[w] ← v
            DFS(w)
            high(v) = min(high(v), high(w))
            if high(w) < DFN[v] then
                count[v]++
    Finished[v] ← true
    F[v] ← count++
    if count[v] > 1 then
        Output: "Not a unique path graph"
        break;
```

2.2 Proof of Correctness

- When either of the above conditions are violated:
  - **Forward/Cross** edge : Existence of multiple paths is easily observed.
  - **Cross** edge : Let $[DFN[u_1], DFN[v_1]]$ and $[DFN[u_2], DFN[v_2]]$ be overlapping such that $DFN[u_1] \leq DFN[u_2]$. Then, we get two paths from v_1 to u_1 -one direct and one via $v_2 \rightarrow u_2$.

  As easily observed, the graph ceases to be a unique path graph when either of these conditions is violated.

- Now, we prove that every unique path graph violates either of these conditions . For this, assume that a graph has (at least) two different paths from v to w. We consider the following cases:
  - **Case 1: w is the ancestor of v** In this case, as observed easily, we can reach v from w either by the normal DFS tree or by taking another path. This another path can either be via another vertex which is a descendant of w or directly to v, necessitating a forward edge (from w to that particular vertex or v, or be via another vertex which is not a descendant of u and hence not an ancestor of v, thus necessitating a cross-edge (from that particular vertex to v).
  - **Case 2: v is the ancestor of w** In this case, the two paths from w to v can be either via another vertices which are the ancestors of u , necessitating a back edge or via other vertices which are not ancestors of u, necessitating a cross edge.
  - **Case 3: w and u don't share an ancestor descendant relationship** In this case, a cross-edge must be present

  Thus, in all the three possible cases, we get that at least one of the three conditions will be violated. Hence, every path violating the unique path proprty will be captured by our graph.

2.3 Time Complexity

This involves DFS for just one vertex and all appended operations (checking for forward and cross edges and ancestors) are $O(1)$ and hence, the time complexity is better than $O(m+n)$ which is what is required.