SOLUTION – DIFFICULT – 180532 & 180653

Algorithm:

First, topologically sort the DAG to get its correct topological order. We need to see all the vertices starting from s and ending at t. Maintain an array count[n], which stores the number of different paths from some vertex v to t in count[v]. We need to assign edge values to all the edges in the DAG such that all the count[s] paths have unique pathids, i.e. unique distance over each of the count[s] paths. To do this, we start from the last vertex, i.e. t and go to the left till we reach the start vertex s and on the way we keep updating the count array as well as the edge values.

For vertex t, set count[t] = 1, for all vertices other than t, $count[v] = \sum_{(v,x) is\ an\ edge} count[x]$

If a vertex does not have any outgoing edges, count value for that vertex is 0. In this way we have stored the number of paths from all the vertices to t in the count array. Now, for a vertex v, we need to set edge weights for all the edges going out from it.

For a vertex v, for all edges (v,x) in E, set (v,x).weight = 0 if count[x] = 0.
Now, define ew = 0. Loop over all edges (v,x) in E,
set (v,x).weight = ew
ew = ew+count[x]

In this way we have set edge weights for all edges in the DAG. This algorithm will result in unique pathids.

PseudoCode:

```
Unique_pathids(G, E, V, s, t){
    G_sorted = Topological_ordering(G);                  //Find topological order of G
    for(v in G_sorted, starting from t and going in reverse){   //Travel the order in reverse
        if(v == t) count[v] = 1;
        else if(v.outdegree == 0) count[v] = 0;
        else{                                            //When v has outgoing edges
            count[v] = 0;
            ew = 0;
            for(egde e = (v,x) in E){
                count[v] = count[v] + count[x];
                if(count[x] == 0) e.weight = 0;
                else{
                    e.weight = ew;
                    ew = ew + count[x];
                }
            }
        }
    }
}
```

Proof of Correctness:

The values stored in the count[v] variable for each vertex v is the no. of paths from v to t and *pathIds* generated have values *0* to *count[v]-1.*

We can prove this using induction on the number of vertices in the graph. For each *v,* considering only the vertices *x* such that *(v,x) in E.* Because we are taking vertices in reverse order of topological sort, it is clear that all *x* come before *v* in the algorithm.

Base Case:
No. of edges is 1 i.e. source and sink are the only vertices in the graph. As is clear, the algorithm assigns edge weight = 0 to *(s,t)*. As no. of paths is 1, our claim is true.

Induction:
Consider a vertex v and all its outgoing edges in the graph. If *v* is removed, we get a DAG with lesser no. of vertices. For this DAG, our hypothesis holds true. Thus, for each (v,x_i) *in E*, $count[x_i]$ stores the no. of all paths from *x* to *t* and each edge is assigned proper weights such that unique *pathId* is generated for each path from x_i to *t*, considering each x_i as source vertex.

Now, for *v*, *count[v]* is no. of paths from *v* to *t* which is the sum of no. of paths from x_i to *t* such for all x_i such that (v,x_i) *in E*. As can be seen in the algorithm, this value is maintained. Now, we know for each x_i *s.t* (v,x_i) *in E*, we have assigned edge weights to get proper *pathIds* considering each one of x_i as source vertex. We have to assign each edge (v,x_i) weights, to get different *pathId* for each path. We maintain a value *ew*, which after each iteration gets incremented by the no. of paths from the current x_i. Each edge is then assigned the new *ew* in the next iteration. It is easy to see that pathId for all paths from x_i to *t* satisfy *pathId* < (v, x_{i+1})*.weight*. Also, the new *pathIds* getting generated for paths from *v* to *t* taking edge (v, x_i) in each iteration also satisfy *pathId* < *ew(of next iteration)*. Thus, each *pathId* is distinct.

For each subsequent x_i we get *pathIds* increasing by 1 as for each x_i, we already have edge weights to generate *pathIds* from *0* to *k-1* where *k* is the no. of paths from x_i to *t* (Induction hypothesis). This ensures that no path gets the same *pathId* and all paths have distinct *pathIds* with range *0* to *count[v]-1*. Thus, this graph also holds the hypothesis.

Thus, by induction, we proved that the algorithm is true for all DAGs.

Thus, at the end the source vertex *s*, *count[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 *patIid* from *0* to $N - 1$ where *N* is number of paths from *s* to *t*.

Time Complexity:

The topological sort takes $O(m+n)$ time. After that starting from the last vertex and going till s while updating the count array will also take $O(m+n)$ time as we go through all vertices and edges only once. Again, setting up the edge weights also requires iterating over every vertex exactly once and going through every edge once, so here also time taken is $O(m+n)$.So, total time complexity is $O(m+n)$.